VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉMY PŘEKLADOVÝCH AUTOMATŮ A JEJICH APLIKACE

SYSTEMS OF TRANSLATION AUTOMATA AND THEIR APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LÍVIA ŽITŇANSKÁ**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ALEXANDER MEDUNA, CSc.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Žitňanská Lívia**
Program: Informační technologie
Název: **Systémy překladových automatů a jejich aplikace**
Systems of Translation Automata and Their Applications
Kategorie: Teoretická informatika

Zadání:

1. Dle instrukcí vedoucího se seznamte s překladovými automaty a gramatickými systémy.
2. Zaveďte systémy překladových automatů analogicky jako gramatické systémy.
3. Dle instrukcí vedoucího studujte vlastnosti těchto systémů. Porovnejte jejich sílu s jinými systémy formálních modelů.
4. Dle pokynů vedoucího uvažujte různé části překladačů. Formalizujte je prostřednictvím těchto systémů.
5. Implementujte a testujte formalizace navržené v bodě 4. Porovnejte je se stávajícími obdobnými softwarovými prostředky.
6. Zhodnoťte dosažené výsledky. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education, 2006, ISBN 0-321-48681-1

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a část 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 27. října 2020

Abstrakt

Cieľom tejto práce je zaviesť jednotný aparát pre lexikálnu a syntaktickú analýzu. Táto jednotka sa nazýva lexikálno-syntaktický prevodník LST a je zložená z konečného prevodníku, zásobníkového prevodníku a komunikačných symbolov. Ich definícia a vlastnosti sú odvodené od komunikačných symbolov zavedených v Paralelne komunikujúcich (PC) gramatických systémoch. Slúžia teda na vzájomnú komunikáciu prevodníkov. Práca sa ďalej zaoberá implementáciou tohto aparátu, popisom implementácie a porovnaním vlastností aparátu s už existujúcimi systémami. Výsledný aparát prekladá nový jazyk AIDA založený na jazyku Python 3 a prvkami z jazyka C na výstupný medzikód.

Abstract

The goal of this thesis is to define uniform unit for lexical and syntactic analysis. This unit is called lexical-syntactic transducer LST and it is composed of finite and pushdown transducer and communication (or query) symbols. Their definition and description are based on communication symbols introduced in parallel communicating grammar systems. Their purpose is to secure communication between those transducers. Thesis also deals with implementation of this unit, contains description of the implementation and comparison of attributes with the already existing systems. The resulting unit translates new language AIDA, which is based on Python 3 with elements from language C, to intermediate code.

Kľúčové slová

konečný prevodník, zásobníkový prevodník, lexikálno-syntaktický prevodník, paralelne komunikujúce gramatické systémy

Keywords

finite transducer, pushdown transducer, lexical-syntactic transducer, parallel communicating grammar systems

Citácia

ŽITŇANSKÁ, Lívia. *Systémy překladových automatů a jejich aplikace*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Systémy překladových automatů a jejich aplikace

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána prof. RNDr. Alexandra Medunu CSs. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Lívia Žitňanská
7.5.2021

Poďakovanie

Rada by som sa poďakovala pánovi prof. RNDr. Alexandrovi Medunovi CSc. za jeho vedenie, čas, pomoc, rady a ochotný prístup. Taktiež by som sa chcela poďakovať mojim rodičom za ich neustálu podporu a obetavosť.

Obsah

Kapitola 1

Úvod

Preklad z hľadiska informatiky možno definovať ako transformáciu zdrojového programu na cieľový program. Týmto procesom, rovnako ako aj jeho komponentami, sa zaoberá teória formálnych jazykov. Ako už z názvu vyplýva, jedným zo zameraní teórie formálnych jazykov je jazyk a jeho štruktúra. Znaky z abecied sa spájajú do slov a tie tvoria jazyk. Na rozdiel od prirodzených jazykov však práca s formálnymi jazykmi vyžaduje konečné prostriedky pre ich špecifikáciu, pretože zatiaľ čo sa konečné jazyky dajú špecifikovať vymenovaním ich slov, pre nekonečné jazyky je táto špecifikácia nemožná. Zavedené teda boli gramatiky a automaty, ktoré dokážu reprezentovať konečné aj nekonečné jazyky konečným spôsobom.

Gramatiky pracujú s terminálmi a neterminálmi, nad ich deriváciami existujú pravidlá. Keďže boli pôvodne definované gramatiky príliš obecné, ich pravidlá boli obmedzené a zaviedlo sa niekoľko typov špeciálnych gramatík, napr. Noam Chomsky v roku 1956 navrhol definíciu štyroch základných typov gramatík. Regulárne a bezkontextové gramatiky majú význam pre definíciu programovacích jazykov.

Koncept gramatík bol rozšírený na štruktúru pozostávajúcu z viacerých gramatických systémov pracujúcich paralelne a komunikujúcich spolu navzájom, pričom všetky prispievajú k tvorbe výsledného reťazca. Tieto gramatiky boli pomenované ako paralelne komunikujúce gramatické systémy, pričom majú skratku PCGS. Vďaka už spomínanej komunikácii a paralelnosti sú PCGS silnejšie, ako bežné jednoduché gramatiky.

Okrem gramatík však možno popísať jazyk aj iným spôsobom - pomocou automatov. Ak možno jazyk definovať zásobníkovým automatom, tak je tento jazyk bezkontextový, teda bol vygenerovaný bezkontextovou gramatikou. O regulárny jazyk vygenerovaný regulárnou gramatikou ide vtedy, keď ho možno definovať vhodným konečným automatom. A čo sú vlastne tieto dva konkrétne automaty a automaty vo všeobecnosti? Automat je algoritmus, ktorý rozhodne, či ľubovoľný reťazec nad vstupnou abecedou patrí alebo nepatrí do daného jazyka. Ako napovedajú názvy, konečný automat je automat s konečným počtom stavov a jeho úlohou je prečítať reťazec nad vstupnou abecedou zľava doprava. Pokiaľ sa dostane do koncového stavu, reťazec akceptuje. Zásobníkový automat okrem vstupnej pásky obsahuje zásobník, ktorý pracuje so symbolmi. Tieto automaty tvoria teoretický model.

Konečný a zásobníkový automat potrebujú jednu úpravu, aby sa v preklade mohli použiť, a to je pridanie výstupnej pásky, na ktorú môžu oba aparáty produkovať výstup. Vstupné aj výstupné pásky možno rozdeliť na štvorce, nad ktorými sa posúvajú čítacia a zapisovacia hlava. Vďaka nim prevodníky čítajú vstup a dokážu zapisovať výstup. Konečný prevodník slúži ako model pre lexikálnu analýzu a zásobníkový automat zas pre syntaktickú analýzu.

Tieto analýzy tvoria prvé dve hlavné fázy prekladu pomocou kompilátorov. Okrem nich

ešte priebieha sémantická analýza, generovanie vnútornej formy programu, optimalizácia a generovanie cieľového kódu. Ako už bolo spomenuté, lexikálna analýza je prvá časť prekladu. Jej úlohou je rozpoznať a klasifikovať lexémy, ktoré potom reprezentuje pomocou tokenov. Vstupom tohto procesu je zdrojový súbor, jeho výstupom je postupnosť tokenov, ktorá sa dostáva na vstup syntaktického analyzátora. Ten určuje, či táto postupnosť patrí do zdrojového jazyka a ak nie, vyžadujeme, aby túto skutočnosť oznámil, prípadne sa z chýb zotavil. Syntaktická analýza zahŕňa viaceré typy syntaktických analyzátorov - zhora-nadol, zdola-nahor či univerzálny typ. Pri syntaktickej analýze zhora-nadol a zdola-nahor sa používajú LL a LR gramatiky. Prvé „L“ znamená, že ide o skenovanie vstupu zľava doprava, druhé písmeno označuje deriváciu, ktorá je konštruovaná touto gramatikou - „L“ pre najľavejšiu a „R“ pre najpravejšiu deriváciu. LL gramatika sa používa pri analýze zhora-nadol, LR pri analýze zdola-nahor.

Keďže syntaktická analýza je zložitá, sú niektoré úlohy, ako odstránenie medzier a komentárov zo zdrojového súboru, vykonávané lexikálnou analýzou. Tým, že sú tieto dve fázy prekladu oddelené, sa zvýši efektivita kompilátoru a je možné pri lexikálnej analýze používať špecifické techniky. Teória formálnych jazykov teda nemá zavedený jednotný aparát, ktorý by tieto dve analýzy popisoval.

Cieľom tejto práce sa stalo zavedenie takého stroja, ktorý by bol schopný vykonávať lexikálnu a syntaktickú analýzu a byť popísaný ako jedna jednotka. Definovaný bol lexikálno-syntaktický prevodník LST, ktorý sa skladá z konečného prevodníku a zásobníkového prevodníku. Ako bolo už spomenuté vyššie, každý z nich vykonáva jednu analýzu, konečný prevodník lexikálnu a zásobníkový prevodník syntaktickú. Oba aparáty však pracujú paralelne a vystupujú ako jedna jednotka. Komunikácia medzi nimi je zabezpečená komunikačnými symbolmi, ktoré boli zavedené na obraz komunikačných symbolov z PC gramatických systémov. Mohli by sme teda zaviesť konečný a zásobníkový prevodník v LST prevodníku analogicky ako gramatické systémy. Práca sa zameriava aj na implementáciu tohto prevodníku.

Nasledujúce kapitoly sa snažia definovať a popísať LST prevodník a jeho implementáciu. Na začiatok sú predstavené paralelne komunikujúce gramatické systémy, pričom sa dôraz kladie na vysvetlenie komunikačných symbolov, keďže tie sú pre LST prevodník kľúčové. Ďalej je priblížená práca konečného a zásobníkového prevodníku, pričom je vysvetlená na jednoduchých príkladoch. Nasleduje predstavenie lexikálnej a syntaktickej analýzy. Kapitola sa taktiež zameriava na rôzne formy tokenov či metódy syntaktickej analýzy a obecne zaužívané postupy pri implementácii. Neskôr je predstavený LST prevodník, popísaná je jeho štruktúra, za čím nasleduje porovnanie jeho fungovania s fungovaním paralelne komunikujúcich gramatických symstémov. Nakoniec je vysvetlená jeho implementácia aj jej odchýlenie od bežných postupov.

Kapitola 2

Základné pojmy

Táto kapitola poskytuje prehľad definícií základných pojmov, ktoré sú v texte tejto práce spomínané a ich pochopenie je zásadné. Predpokladá sa základná znalosť teórie formálnych jazykov. Definície boli čerpané z [7] a [6].

2.1 Abeceda, reťazec, jazyk

Táto sekcia čerpá definície z [7].

Definícia 2.1.1. Abeceda je ľubovoľná neprázdna konečná množina. Prvky abecedy sa nazývajú symboly.

Definícia 2.1.2. Nech je Σ abeceda. **Reťazec** nad touto abecedou je konečná postupnosť symbolov z Σ. Prázdnou postupnosťou nazývame prázdne slovo a označujeme ho ϵ.

Pri zápise reťazcov sa vynechávajú čiarky, takže reťazec a,b,c,d,e nad abecedou Σ sa pre jednoduchosť bude zapisovať *abcde*.

Definícia 2.1.3. Nech x je reťazec nad abecedou Σ. **Reverzácia** reťazca x, $reversal(x)$, je definovaná:

1. ak $x = \epsilon$ potom $reversal(\epsilon) = \epsilon$
2. ak $x = a_1...a_n$ potom $reversal(a_1...a_n) = a_n...a_1$ pre $n \geq 1$ a $a_i \in \Sigma$ pre všetky $i = 1, .., n$

Definícia 2.1.4. Nech x a y sú dva reťazce nad abecedou Σ; x je **prefixom y**, ak existuje reťazec z nad abecedou Σ, pričom platí $xz = y$.

Definícia 2.1.5. Nech x a y sú dva reťazce nad abecedou Σ; x je **sufixom y**, ak existuje reťazec z nad abecedou Σ, pričom platí $zx = y$.

Definícia 2.1.6. Nech x a y sú dva reťazce nad abecedou Σ; x je **podreťazcom y**, ak existujú reťazce z, z' nad abecedou Σ, pričom platí $zxz' = y$.

Definícia 2.1.7. Nech je Σ abeceda a nech $L \subseteq \Sigma^*$, pričom Σ^* značí podmnožinu všetkých reťazcov nad Σ. Potom L je **jazyk** nad touto abecedou. Prázdna množina $\emptyset$ a $\{\epsilon\}$ sú teda jazyky nad každou abecedou. Platí však $\emptyset \neq \{\epsilon\}$

2.2 Gramatika a jej derivácie

Definícia 2.2.1. Bezkontextová gramatika je štvorica $G = (N, T, P, S)$ kde

N je abeceda

T je abeceda taká, že $N \cap T = \emptyset$

$P \subseteq (N \cup T)^* N (N \cup T)^* \mathrm{x} (N \cup T)^*$ je konečná množina

$S \in N$

Pričom N nazývame abecedu neterminálov, T abecedu terminálov, P konečnú množinu pravidiel a S počiatočný neterminál.

Pravidlo $(x, y) \in P$ obvykle zapisujeme: $x \Rightarrow y$.

Definícia 2.2.2. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Nech $u, v \in (N \cup T)^*$ a $p = A \to x \in P$. Potom uAv **priamo derivuje** uxv za použitia p v G, zapísané $uAv \Rightarrow uxv[p]$ alebo zjednodušene $uAv \Rightarrow uxv$. Ak $uAv \Rightarrow uxv$ v G, môžeme povedať, že G vykonáva **derivačný krok** z uAv do uxv. [6]

Počas najľavejšieho derivačného kroku je prepísaný najľavejší neterminál.[6]

Definícia 2.2.2.1. Nech $G = (N, T, P, S)$ je bezkontextová gramatika, nech $u \in T^*$,$v \in (N \cup T)^*$, $p = A \to x \in P$ je pravidlo. Potom uAv priamo derivuje uxv pomocou **najľavejšej derivácie** použitím pravidla p v G, zapísané ako $uAv \Rightarrow_{lm} uxv[p]$

Počas najpravjšieho derivačného kroku je prepísaný najpravší neterminál.[6]

Definícia 2.2.3. Nech $G = (N, T, P, S)$ je bezkontextová gramatika, nech $u \in (N \cup T)^*$,$v \in T^*$, $p = A \to x \in P$ je pravidlo. Potom uAv priamo derivuje uxv pomocou **najpravšej derivácie** použitím pravidla p v G, zapísané ako $uAv \Rightarrow_{rm} uxv[p]$

Definícia 2.2.4. Každé slovo, ktoré možno odvodiť z počiatočného symbolu danej gramatiky, nazývame **vetná forma.**

Zhrnutie

Táto sekcia sa venovala pripomenutiu zopár základných pojmov, ktoré sa v práci vyskytujú. Od čitateľa sa očakáva základná znalosť teórie formálnych prekladačov, no napriek tomu mu boli niektoré pripomenuté.

Kapitola 3

Paralelne komunikujúce gramatické systémy

Pralelne komunikujúce gramatické systémy, skrátene PC gramatické systémy alebo PCGS, sú systémy, ktoré rozširujú koncept gramatiky do štruktúry umožňujúcej paralelné pracovanie viacerých gramatík súčasne a prispievanie do produkcie reťazcov. V týchto systémoch je jedna komponenta nadradená, nazýva sa *master*, ostatné komponenty sú pomocné – *slave* (sluhovia). Všetky tieto komponenty sa podieľajú na derivácii a ovplyvňujú výsledný reťazec, ktorý systém vyprodukuje. Deriváciu kontroluje gramatika *master* a je považovaná za dokončenú, keď vyprodukuje reťazec terminálov. Na stav reťazcov v ostatných komponentách sa neberie ohľad. [9] [10]

Definícia 3.1. Paralelne komunikujúci gramatický systém stupňa n, kde $n \leq 1$, je konštrukcia:[9]

$$\Gamma = (N, K, T, (S_1, P_1), ..., (S_n, P_n)) \text{ , kde}$$

- N je množina neterminálov
- K je konečná množina komunikačných symbolov, $K = \{Q_1, ..., Q_n\}$
- T je množina terminálov
- P_i je konečná množina pravidiel v tvare
  $$A \to x$$
  s $A \in N$ a $x \in (N \cup T \cup K)^*$, pre všetky $i = 1, ..., n$
- S_i je počiatočný symbol i-tej komponenty, $S_i \in N$ pre všetky $i = 1, ..., n$

Na to, aby sa aj pomocné komponenty *slave* podieľali na derivácii, bolo nutné zaviesť komunikačné kroky. Princíp ich funkcie pozostáva z toho, že rôzne komponenty v systéme môžu vzájomne zdieľať reťazce - gramatika uvedie vo svojom obsahu požiadavku na reťazec inej gramatiky. Všetky prepisujúce kroky sú pozastavené dovtedy, kým nie je komunikácia dokončená, teda dokým gramatika neobdrží reťazec z požadovanej komponenty.
Gramatiky komunikujú v jednom z dvoch typov derivácii:

- generatívnej
- komunikačnej

Vždy sú uprednostňované komunikačné kroky pred generatívnymi.

Generatívny krok prebieha buď rovnako ako v bezkontextových gramatikách, teda reťazec x_i priamo derivuje reťazec y_i v gramatike G_i, alebo ak sú oba reťazce identické musia byť v množine terminálov. Pri komunikačnom kroku sa najskôr zistí, či reťazec x_i obsahuje komunikačný symbol. Ak obsahuje a súčasne sa v komponente x_j, ktorej reťazec požadujeme, nenachádza žiaden komunikačný symbol, vkopíruje sa hodnota reťazca z komponenty x_j do komponenty x_i.

Definícia 3.2. *Generatívny krok:*

- buď $x_i \Rightarrow y_i$ v $G_i = (N \cup K, T, P_i, S_i)$,
- alebo $x_i = y_i \in T*$

pre všetky $1 \leq i \leq n$, potom

$$(x_1, ..., x_n) \Rightarrow_g (y_1, ..., y_n)$$

Definícia 3.3. *Komunikačný krok:*

- množina $z_i = x_i$ pre všetky $i = 1, ..., n$

Pre všetky $i = 1, ..., n$ ak

$$\text{alph}(x_i) \cap K \neq \emptyset$$

a pre každé Q_j v x_i

$$\text{alph}(x_j) \cap K = \emptyset$$

potom pre každé Q_j v x_i

1. nastav $z_j = S_j$,
2. zameň Q_j za x_j v x_i,
3. nastav z_i do reťazca z bodu (2)

Vykonaj

$$(x_1, ..., x_n) \Rightarrow_c (y_1, ..., y_n)$$

s $y_i = z_i$, pre všetky $i = 1, ..., n$

Ak možno vykonať generatívny krok

$$(x_1, ..., x_n) \Rightarrow_g (y_1, ..., y_n)$$

alebo komunikačný krok

$$(x_1, ..., x_n) \Rightarrow_c (y_1, ..., y_n)$$

potom výsledný krok zapisujeme

$$(x_1, ..., x_n) \Rightarrow (y_1, ..., y_n)$$

Generovaný jayzk $L(\Gamma) = \{x \in T^* : (S_1, S_2, ..., S_n) \Rightarrow^* (x, \alpha_2, ..., \alpha_n), \alpha_i \in (N \cup T \cup K)^*,$ pre všetky $i = 2, ..., n\}$

Príklad 3.4. Nasledujúci príklad je prebratý z [9].

Majme $\Gamma = (\{S_1, S_1^{'}, S_2, S_3\}, K, \{a, b, c\}, (S_1, P_1), (S_2, P_2), (S_3, P_3))$, kde

$P_1 = \{S_1 \to abc, S_1 \to a^2b^2c^2, S_1 \to aS_1^{'}, S_1 \to a^3Q_2, S_1^{'} \to aS_1^{'}, S_1^{'} \to\ a^3Q_2, S_2 \to b^2Q_3, S_3 \to c\}$
$P_2 = \{S_2 \to bS_2\}$
$P_3 = \{S_3 \to cS_3\}$

$$\begin{aligned}(S_1, S_2, S_3) &\Rightarrow (aS_1^{'}, bS_2, cS_3) \Rightarrow^* (a^nS_1^{'}, b^nS_2, c^nS_3)\\ &\Rightarrow (a^{n+3}Q_2, b^{n+1}S_2, c^{n+1}S_3) \Rightarrow (a^{n+3}b^{n+1}S_2, S_2, c^{n+1}S_3)\\ &\Rightarrow (a^{n+3}b^{n+3}Q3, bS_2, c^{n+2}S_3) \Rightarrow a^{n+3}b^{n+3}c^{n+2}S_3, bS_2, S_3)\\ &\Rightarrow (a^{n+3}b^{n+3}c^{n+3}S_3, bbS_2, cS_3)\end{aligned}$$

Výsledný vygenerovaný jazyk:

$$L_r(\Gamma) = L_{nr}(\Gamma) = \{a^nb^nc^n : n \leq 1\}$$

Zhrnutie

Jedno z rozšírení konceptu gramatík dalo za vznik paralelne komunikujúcim gramatickým systémom, PCGS. V tejto kapitole sme ich predstavili, rozdelili na *master* a *slave* komponenty a vysvetlili si rozdiely medzi nimi. V neposlednom rade sme sa zamerali na generatívny a komunikačný krok, komunikačné symboly a na príklade sme ukázali komunikáciu medzi komponentami PCGS aj stredanie generatívnych a komunikačných krokov.

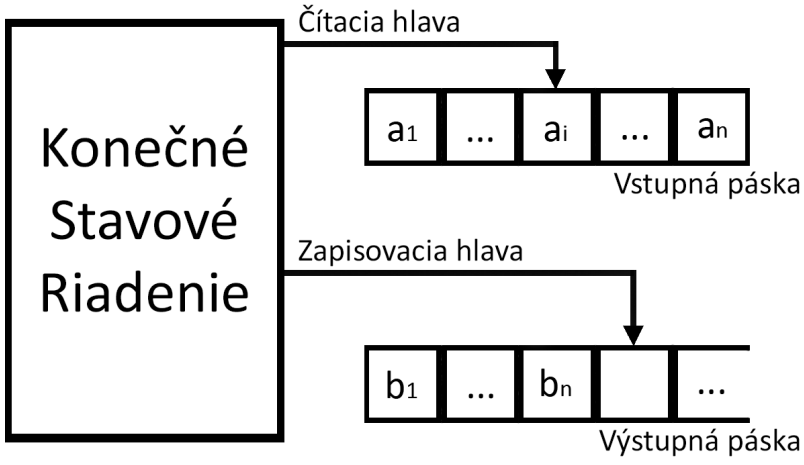
Kapitola 4

Konečný a zásobníkový prevodník

Nasledujúca kapitola sa venuje konštrukciám konečný a zásobníkový prevodník, z ktorých vychádza štruktúra lexikálno-syntaktického prevodníku. Kapitola obsahuje stručný úvod ku každej konštrukcii, ich definície a vlastnosti. Znalosť o fungovaní konečného a zásobníkového prevodníka je kľúčová pre pochopenie nasledujúcich kapitol. [5], [8], [2]

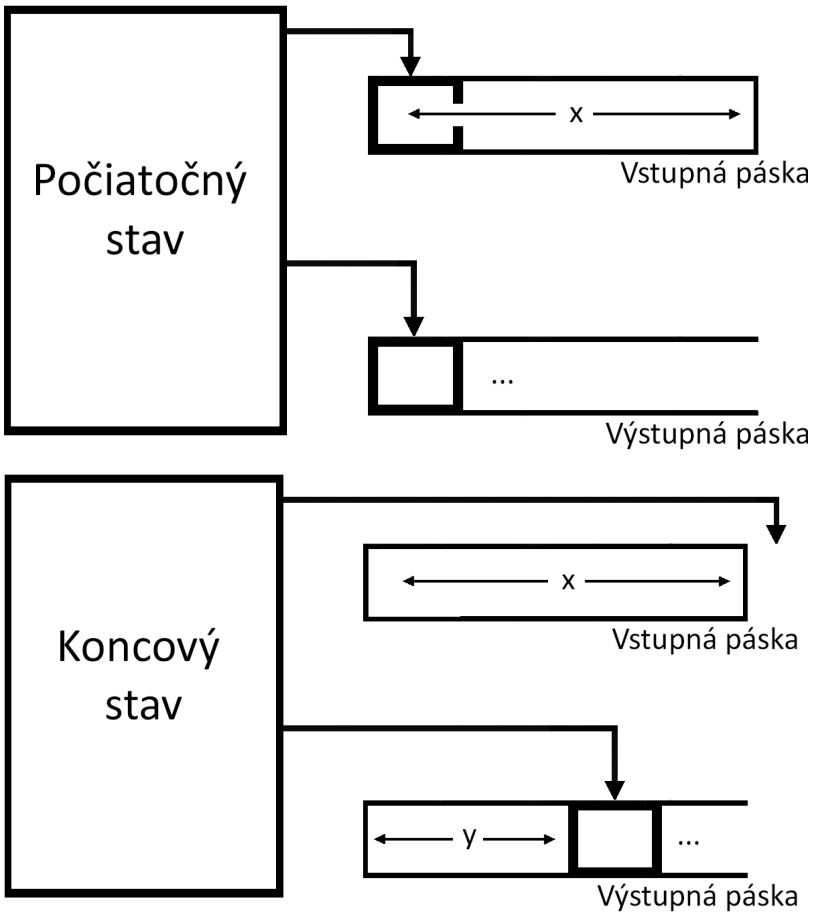
4.1 Konečný prevodník

Najjednoduchší prevodník je konečný prevodník, ktorý vznikol rozšírením konečného automatu tak, aby mohol čítať a taktiež zapisovať. Konečný prevodník M, sa skladá zo vstupnej pásky, ktorá slúži len na čítanie, a výstupnej pásky, ktorá slúži len na zápis. Obe pásky možno rozdeliť na štvorce, kde každý štvorec obsahuje jeden symbol zo zadaného vstupného reťazca, $a_1....a_i....a_n$. Aktuálny vstupný symbol je reprezentovaný symbolom a_i, ktorý je naskenovaný čítacou hlavou zo vstupnej pásky. Výstupnú pásku možno charakterizovať ako polo-nekonečnú - jej štvorce sa môžu nekonečne predlžovať doprava. Ak je výstupná páska naplnená reťazcom $b_1...b_n$, potom sa zapisovacia hlava vyskytuje nad štvorcom, ktorý nasleduje za štvorcom obsahujúcim b_n. [5], [2]



Obrázok 4.1.1. Reprezentácia konečného prevodníku. Preložené a prevzaté z [5].

Konečné stavové riadenie je reprezentované konečnou množinou stavov spolu s konečnou reláciou, ktorá je špecifikovaná ako množina výpočtových pravidiel. Prevodník M pracuje tak, že vykonáva posuny podľa jeho výpočtových krokov. Ak zo vstupnej pásky prevodník neprečíta žiaden symbol, čítacia hlava zostane na mieste. Ak zo vstupnej pásky prečíta symbol a_i, čítaciu hlavu posunie o jeden štvorec doprava a zapisovaciu hlavu posunie o štvorec za výstupný reťazec. Konečný prevodník obsahuje stavy, z ktorých je jeden definovaný ako počiatočný stav, a niektoré sú označené ako koncové stavy.
Ak vstupná páska obsahuje reťazec x a výstupná páska je prázdna, prevodník môže začať preklad reťazca x z počiatočného stavu.Preklad x na y konečným prevodníkom M nastáva vtedy, keď M spraví postupnosť krokov takú, že všetky x zo vstupnej pásky sú prečítané a na výstupnú pásku je zapísaný reťazec y a prevodník skončí v koncovom stave. Množina obsahujúca všetky páry reťazcov preložených týmto spôsobom formuje preklad definovaný prevodníkom M.[5]



Obrázok 4.1.2. Preklad x na y pomocou konečného prevodníku. Prebraté z [5].

Definícia 4.1.3. M je konečný prekladový automat $M = (Q, \Sigma, R, s, F)$ [5], kde

- Q je konečná množina stavov
- Σ je abeceda taká, že $\Sigma \cap Q = \emptyset$ a $\Sigma = \Sigma_I \cup \Sigma_O$, kde Σ_I je vstupná abeceda a Σ_O je výstupná abeceda

- $R \subseteq Q(\Sigma_I \cup \{\epsilon\}) \times Q\Sigma_O^*$ je konečná relácia
- $s \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

Definícia 4.1.4. Výpočtový krok [5]
Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. Objekty patriace R sa nazývajú pravidlá a R odkazuje na konečnú množinu pravidiel. Uvažujme pravidlo $(pa, qz) \in R$ kde $p, q \in Q, a \in \Sigma_I \cup \{\epsilon\}$, a $z \in \Sigma_O^*$, namiesto (pa, qz) sa zapisuje:

$$r : pa \Rightarrow qz$$

skrátene $pa \Rightarrow qz$

Príklad 4.1.5. Príklad je inšpirovaný z [2].
Nech existuje konečný prevodník $M = (Q, \Sigma, R, s, F)$ kde:

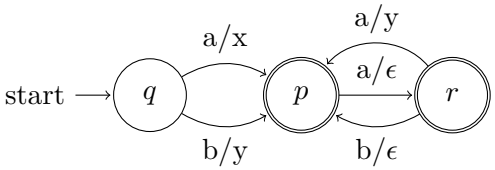
- $Q = \{q, p, r\}$
- $\Sigma_I = \{a, b\}$
- $\Sigma_O = \{x, y\}$
- $R = \{qa \Rightarrow px, qb \Rightarrow px, pb \Rightarrow ry, pa \Rightarrow r\epsilon, ra \Rightarrow py, rb \Rightarrow p\epsilon\}$
- $s = q$
- $F = \{p, r\}$

Majme počiatočný reťazec aaa. Konečný prevodník potom vykoná nasledujúcu postupnosť krokov:

$$\mathbf{q}aaa \Rightarrow \mathbf{p}x$$
$$\mathbf{p}aa \Rightarrow \mathbf{r}x$$
$$\mathbf{r}a \Rightarrow \mathbf{r}xy$$

Konečný prevodník z príkladu prijíma symboly a a b a prekladá ich na symboly x a y tak, že výstupný reťazec má tvar x^1y^*.

V grafickom znázornení prevodníku z príkladu 4.1.5, ktorý nasleduje, si možno všimnúť rozdiel medzi konečným automatom a prevodníkom - po vstupnom symbole sa za oddeľovacím znakom / nachádza výstupný symbol, napríklad a/x, kde a je vstupný symbol a x je symbol výstupnej abecedy. Graficky by bol tento konečný prevodník znázornený nasledovne:



Obrázok 4.1.5.1. Grafické znázornenie prevodníku z príkladu 4.1.5

Definícia 4.1.6. Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. **Vstupný konečný automat** M_I je definovaný ako [5]

$$M_I = (Q, \Sigma_I, R_I, s, F)$$

kde Σ_I je vstupná abeceda M, a

$$R_I = \{qa \to p : qa \Rightarrow px \in R \text{ pre niektoré } x \in \Sigma_O^*\}$$

Definícia 4.1.7. Konfiguráciu M definujeme ako trojicu (q, x, y): [2]

- $q \in Q$ je aktuálny stav konečného riadenia
- $x \in \Sigma_I^*$ je vstupný reťazec, ktorý zostal na vstupnej páske a čítacia hlava sa nachádza nad najľavejším symbolom z x
- $y \in \Sigma_O^*$ je výstupný reťazec dosiaľ zapísaný

Konfigurácia konečného prevodníku M is reťazec χ, ktoré má formu [5]

$$\chi = \chi_I | y$$

kde χ_I je konfigurácia vstupného konečného automatu M_I. | je špieciálny symbol ($| \notin \Sigma$), a $y \in \Sigma_O^*$. Pokiaľ máme konečný prevodník M, jeho konfigurácia sa označuje ako χ_M. Toto označenie však možno zjednodušiť na χ.

Definícia 4.1.8. Krok konečného prevodníku [5]
Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. Taktiež, nech

$$\chi = \chi_I | y \text{ a } \chi^{'} = \chi^{'} | yz$$

sú dve konfigurácie M, a nech $r : qa \Rightarrow pz \in R$. Ak

$$\chi \vdash \chi_I \qquad [qa \vdash p]$$

vo vstupnom konečnom automate M_I, potom M vykonáva krok z χ do $\chi^{'}$ podľa r, symbolicky sa zapisuje ako

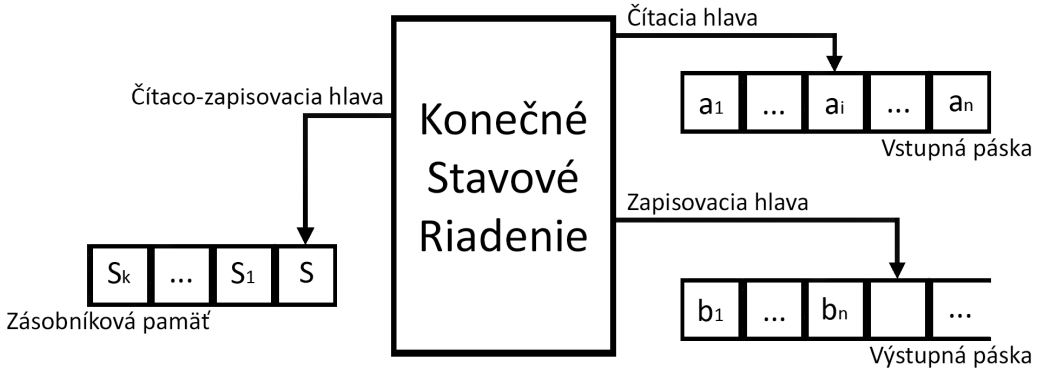
$$\chi \Vdash \chi^{`} \qquad [r] \qquad .$$

4.2 Zásobníkový prevodník

Zásobníkový prevodník M je založený na zásobníkovom automate. Obsahuje vstupnú pásku a výstupnú pásku a na rozdiel od konečného prevodníku obsahuje aj zásobníkovú pamäť, ktorej vrchol ovplyvňuje každý prechod týmto prevodníkom vykonaný. Rovnako ako konečný prevodník, vstupnú aj výstupnú pásku zásobníkového prevodníku možno rozdeliť na štvorce. Každý štvorec vstupnej pásky obsahuje jeden symbol zo vstupného reťazca $a_1.....a_i....a_m$, kde sa čítacia hlava nachádza nad aktuálnym symbolom, a_i. Výstupnú pásku

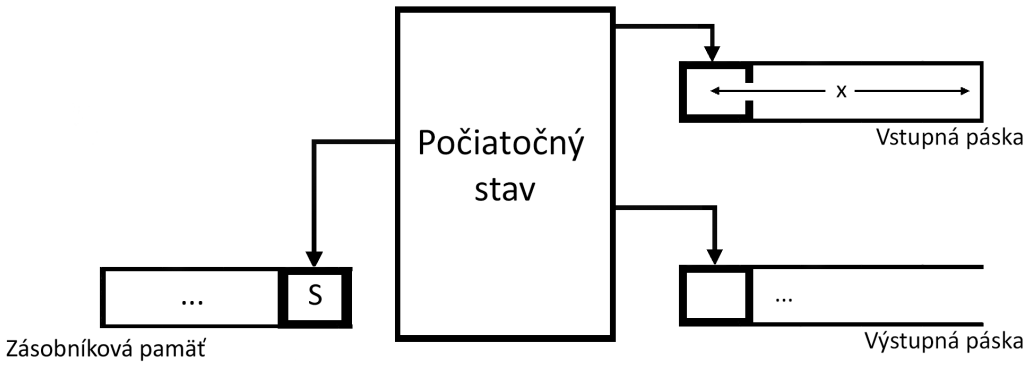
možno charakterizovať ako polo-nekonečnú - jej štvorce sa môžu nekonečne predlžovať doprava. Ak obsahuje reťazec $b_1...b_n$, tak sa zapisovacia hlava nachádza nad najpravším symbolom b_n. Zásobník má v tomto prevodníku rovnakú úlohu, ako pri zásobníkovom automate.

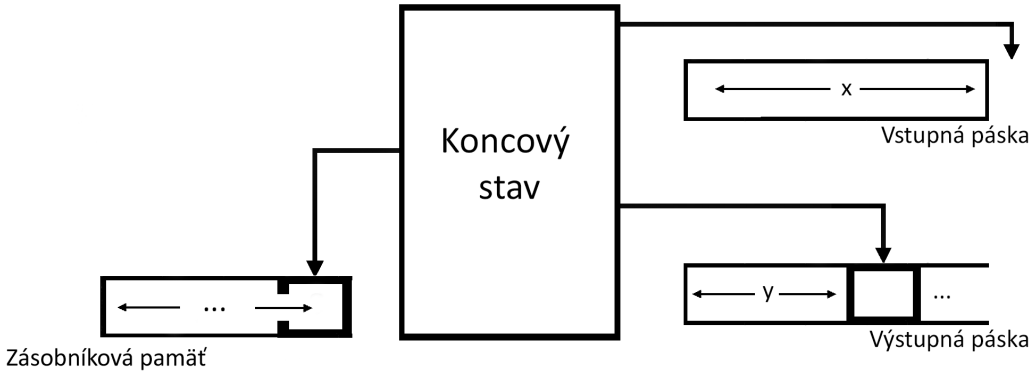
Konečné riadenie prevodníka M je reprezentované konečnou množinou stavov spolu s konečným vzťahom, obvykle špecifikovaným konečnou množinou pravidiel. Zo stavov prevodníku M je jeden definovaný ako počiatočný, niektoré ako koncové. Pre zásobníkový prevodník sú zadané pravidlá, podľa ktorých vykonáva pohyby. Pohyb začína zmenou aktuálneho stavu, pokračuje výmenou najvrchnejšieho symbolu zásobníka reťazcom, prečítaním žiadneho alebo jedného vstupného symbolu zo vstupnej pásky a zápisu výsledného reťazca.[5]



Obrázok 4.2.1. Reprezentácia zásobníkového prevodníku. Obrázok je prebratý z [5].

Počiatočný stav zásobníkového prevodníku je definovaný vstupnou páskou obsahujúcou reťazec x, zásobníkom obsahujúcim počiatočný symbol S a prázdnou výstupnou páskou. Preklad x na y zásobníkovým prevodníkom M je proces, pri ktorom sa z počiatočného stavu prevodník dostane do konečného tým, že prečíta celý reťazec x zo vstupnej pásky a na výstupnú pásku zapíše reťazec y.[5]





Obrázok 4.2.2. Preklad x na y pomocou konečného prevodníku.Obrázok je prebratý z [5].

Definícia 4.2.3. M je zásobníkový prekladový automat $M = (Q, \Sigma, R, s, F)$, kde [5]

- Q je konečná množina stavov
- Σ je abeceda taká, že $\Sigma \cap Q = \emptyset$ a $\Sigma = \Sigma_I \cup \Sigma_O \cup \Gamma$,kde Σ_I je vstupná abeceda a Σ_O je výstupná abeceda a Γ je zásobníková abeceda obsahujúca počiatočný symbol S
- $R \subseteq \Gamma Q(\Sigma_I \cup \{\epsilon\}) \times \Gamma^* Q \Sigma_O^*$ je konečná relácia
- $s \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

Uvažujme zásobníkový prevodník $M = (Q, \Sigma, R, s, F)$. Objekty patriace R sa nazývajú pravidlá a R odkazuje na konečnú množinu pravidiel. Nech existuje pravidlo $(Apa, wqv) \in R$ kde $A \in \Gamma, p, q \in Q, a \in \Sigma_I \cup \{\epsilon\}, w \in \Gamma^*$, a $z \in \Sigma_O^*$, namiesto (Apa, qv) sa zapisuje:

$$r : Apa \Rightarrow uqv$$

skrátene $Apa \Rightarrow uqv$

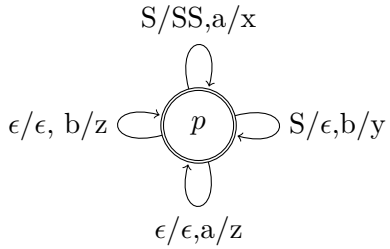
Príklad 4.2.4. Nech existuje zásobníkový prevodník $M = (Q, \Sigma, R, s, F)$ kde:

- $Q = \{p\}$
- $\Sigma_I = \{a, b\}$
- $\Sigma_O = \{x, y, z\}$
- $\Gamma = \{S\}$
- $R = \{Spa \Rightarrow SSpx, Spb \Rightarrow \epsilon py, \epsilon pa \Rightarrow \epsilon pz, \epsilon pb \Rightarrow \epsilon pz\}$
- $s = p$
- $F = \{p\}$

Ak bude vstupný reťazec *abbab*, prevodník vykoná nasledujúcu postupnosť krokov:

$S\mathbf{p}abbab \Rightarrow SS\mathbf{p}x$
$SS\mathbf{p}bbab \Rightarrow S\mathbf{p}xy$
$S\mathbf{p}bab \Rightarrow \mathbf{p}xyy$
$\mathbf{p}ab \Rightarrow \mathbf{p}xyyz$
$\mathbf{p}b \Rightarrow \mathbf{p}xyyzz$

Zásobníkový prevodník z príkladu prijíma symboly a a b a prekladá ich na symboly x, y a z vzhľadom na obsah zásobníkovej pamäte. V tomto prípade bude reťazec *abbab* preložený na reťazec $xyyzz$. Príklad bol inšpirovaný [2].



Obrázok 4.2.5. Grafické znázornenie zásobníkového prevodníku z príkladu 4.2.4.

Možno si povšimnúť, že na rozdiel od zásobníkového automatu obsahujú prechody symbol / oddeľujúci symboly vstupnej abecedy Σ_I a zásobníkovej abecedy Γ od symbolov z výstupnej abecedy Σ_O. Napríklad prechod označený $S/SS, a/xS$ obsahuje symboly S a SS, čo sú symboly zásobníkovej abecedy, kde S je zo zásobníka vybraný a SS sú 2 znaky na zásobník vložené. a je symbol zo vstupnej abecedy a x je znak z výstupnej abecedy.

Definícia 4.2.6. Vstupný zásobníkový automat Nech $M = (Q, \Sigma, R, s, F)$ je zásobníkový prevodník. Vstupný zásobníkový automat M_I je definovaný ako [5]

$$M_I = (Q, \Sigma_I \cup \Gamma, R_I, s, F)$$

kde Σ_I je vstupná abeceda M, Γ je zásobníková abeceda prevodníka M a

$$R_I = \{Aqa \to up : Aqa \Rightarrow px \in R \text{ pre niektoré } x \in \Sigma_O^*\}$$

Definícia 4.2.7. Konfiguráciu M definujeme ako štvoricu (q, x, a, y):[2]

- $q \in Q$ je aktuálny stav konečného riadenia
- $x \in \Sigma_I^*$ je vstupný reťazec, ktorý zostal na vstupnej páske a čítacia hlava sa nachádza nad najľavejším symbolom z x
- $\alpha \in \Gamma$ je obsah zásobníkovej pamäte, kde najľavejší symbol α je súčasne najvrchnejším symbolom. Ak je $\alpha = \epsilon$, tak je prádzna aj zásobníková páska.
- $y \in \Sigma_O^*$ je výstupný reťazec dosiaľ zapísaný

Konfiguráciu prevodníka M má formu [5]

$$\chi = \chi_I | y$$

kde χ_I je konfigurácia vtupného konečného automatu M_I, kde | je špeciálny symbol ($| \notin \Sigma$), a $y \in \Sigma_O^*$. Ak máme prevodník M, jeho konfigurácia je označená ako χ_M. Tento zápis možno upraviť, teda χ_M sa zjednoduší na χ.

Definícia 4.2.8. Krok zásobníkového automatu [5]
Nech $M = (Q, \Sigma, R, s, F)$ je zásobníkový prevodník. Taktiež, nech

$$\chi = \chi_I | y \text{ a } \chi^{'} = \chi^{'} | yz$$

sú dve konfigurácie M, a nech $r : Aqa \Rightarrow upz \in R$. Ak

$$\chi \vdash \chi_I \qquad [Aqa \vdash up]$$

vo vstupnom konečnom automate M_I, potom M vykonáva krok z χ do $\chi^{'}$ podľa r, symbolicky sa zapisuje ako

$$\chi \Vdash \chi^{'} \qquad [r] \qquad .$$

Ak $\delta(q, \alpha, Z)$ obsahuje (r, α, z), potom zapisujeme $(q, ax, Z\gamma, y) \Rightarrow (r, x, \alpha\gamma, yz)$ pre všetky $x \in \Sigma_I^*, \gamma \in \Gamma^*)$, a $y \in \Sigma_0^*$.

Zhrnutie

Konečný a zásobníkový prevodník vznikli pridaním výstupnej pásky ku konečnému a zásobníkovému automatu. V tejto kapitole bolo vysvetlené ich fungovanie, pohyb čítacej, zapisovacej, v prípade zásobníkového prevodníku, aj čítaco-zapisovacej hlavy počas prekladu. Definície boli obohatené grafickými znázorneniami a princíp činnosti týchto aparátov bol demonštrovaný na jednoduchých príkladoch.

Kapitola 5

Lexikálna a syntaktická analýza

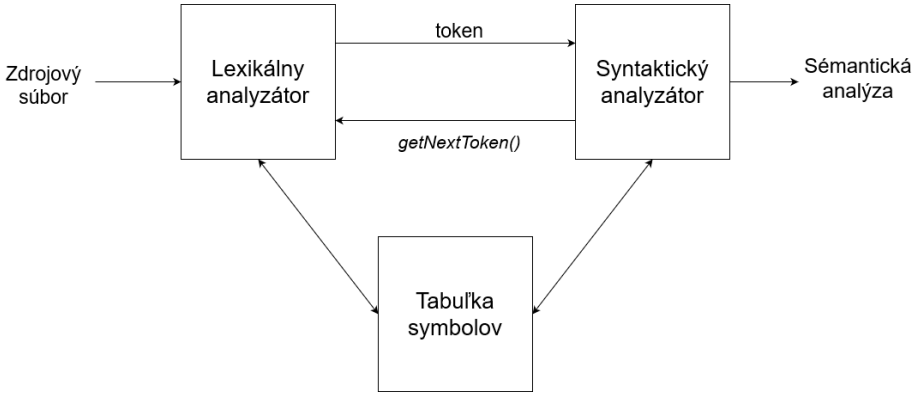
Kompilátor je program, ktorý na vstupe prijíma zdrojový program zapísaný v zdrojovom jazyku, na výstup k nemu generuje funkčne ekvivalentný cieľový program v cieľovom jazyku. Preklad je tranformácia zdrojového jazyka na cieľový pomocou kompilátoru. [7] Preklad prebieha v šiestich fázach:

- Lexikálna analýza
- Syntaktická analýza
- Sémantická analýza
- Generovanie vnútornej formy programu
- Optimalizácia
- Generovanie cieľového kódu

Lexikálna a syntaktická analýza sú prvé dve hlavné fáze prekladu, ktorých implementácia je zásadná pre aplikáciu vytvorenú v tejto práci. Ich pochopenie je kľúčové a preto sa im nasledujúca kapitola bude venovať. [1], [6], [3]

5.1 Lexikálna analýza

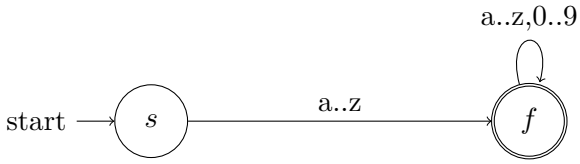
Prvá časť kompilátoru sa nazýva lexikálna analýza. Zaoberá sa zoskupovaním reťazcov zo zdrojového programu do postupnosti lexikálnych symbolov, ktorá bude poslaná na syntaktickú analýzu, teda rozpoznáva a klasifikuje lexémy a reprezentuje ich pomocou tokenov. Lexikálna analýza často pracuje s tabuľkou symbolov, používa ju na ukladanie lexémov figurujúcich ako identifikátory. Takáto lexéma býva prečítaná lexikálnym analyzérom - scannerom (skenerom) ako výpomoc syntaktickému analyzátoru - parseru. Zvyčajne táto interakcia býva implementovaná vo funkcii *getNextToken()* tak, že parser volá lexikálny analyzátor, pričom toto volanie zapríčiní, že lexikálny analyzátor číta znaky zo vstupného súboru dovtedy, kým ich nemožno identifikovať ako tokeny, ktoré potom predá parseru.[7] [1]



Obrázok 5.1.1. Interakcia medzi lexikálnym a syntaktickým analyzátorom. Prevzaté z [1].

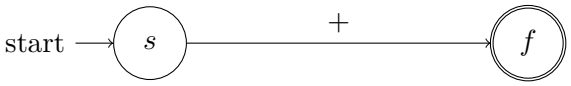
Keďže lexikálna analýza pracuje priamo so vstupným súborom, okrem identifikácie lexémov medzi jej úlohy patrí odstraňovanie komentárov či medzier, nových riadkov, tabulátorov, prípadne iných znakov, ktoré oddeľujú tokeny vo vstupnom súbore. Okrem toho však môže lexikálny analyzátor počítať nové riadky a na základe toho vyhodnocovať, na ktorých riadkoch nastala chyba. Pokiaľ zdrojový súbor používa makro-procesor, rozšírenie makier môže byť taktiež úlohou lexikálneho analyzátoru. [1] [7]

Rozpoznávanie jednotlivých lexémov prebieha v deterministickom konečnom automate:



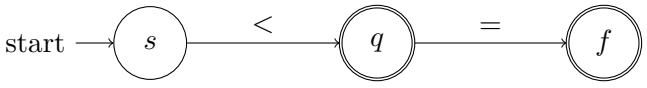
Obrázok 5.1.2. Prijímanie identifikátoru deterministickým automatom. Prevzaté z [6].

Vyššie uvedený graf zobrazuje prijímanie identifikátoru. Z počiatočného stavu s sa načítaním vstupného alfanumerického znaku presunie do koncového stavu f, kde pokračuje v načítavaní alfanumerických a tento raz aj numerických znakov.



Obrázok 5.1.3. Prijímanie operátoru + deterministickým automatom. Prevzaté z [6].

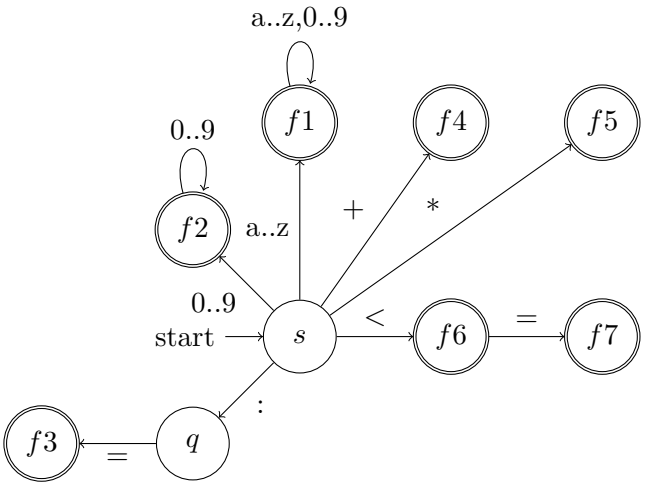
Nad koncovým stavom f sa nenachádza slučka, pretože na rozdiel od predošlého príkladu, operátor plus je len jeden znak, kdežto identifikátor môže obsahovať viac znakov.



Obrázok 5.1.4. Prijímanie operátoru $\leq$ deterministickým automatom. Prevzaté z [6].

Keďže operátor $\leq$ pozostáva presne z dvoch znakov, nie je potrebné zavádzať slučku nad koncovým stavom.

Bežne sa lexikálny analyzátor implementuje ako jeden deterministický stavový automat, teda všetky príklady uvedené vyššie majú rovnaký počiatočný stav s, koncové stavy sú však individuálne pre každý typ tokenu. Taký deterministický konečný automat by vyzeral nasledovne:



Obrázok 5.1.5. Prijímanie lexémov deterministickým automatom [6]

Graf 5.1.5 zobrazuje deterministický konečný automat, ktorý prijíma identifikátory, celé čísla, operátor priradenia :=, ďalej operátory +,∗,< a $\leq$. Stav s je počiatočný stav a stavy $f1, f2, f3, f4, f5, f6$ a $f7$ sú koncové stavy. Práve v týchto stavoch sa na základe predošlých vstupných stavov vytvoria tokeny.
Nasledujúci kúsok kódu zobrazuje implementačnú metódu na určenie typu lexému:

Algoritmus 1: Určenie typu lexému

```
while a je ďalší znak znak zdrojového programu and M môže vykonať prechod so
 symbolom a do
    Prečítaj znak a
    Vykonaj prechod
    if M je v koncovom stave then
        Urči typ lexému, ktorý korešponduje danému stavu
    end
    else
        Nahlás lexikálnu chybu
    end
end
```

5.1.6 Tokeny a ich forma

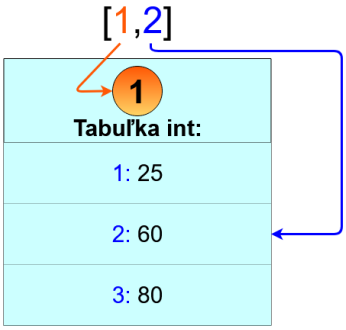
Token, zmienený v 5.1, reprezentuje jednotlivo každý lexém zo zdrojového programu. Obecne sa token skladá z dvoch položiek - jeho typu a atribútu:

[**type**, *attribute*]

Je to teda pár pozostávajúci z názvu tokenu a hodnoty, ktorá nie je povinná. Názov (typ) tokenu je abstraktný symbol, ktorý reprezentuje nejakú lexikálnu jednotku, napríklad kľúčové slovo alebo sekvencia vstupných znakov značiacich identifikátor ako v 5.1.2. Sú to vstupné symboly, ktoré spracováva parser. Token sa často označuje podľa svojho mena. Atribúty tokenu slúžia na uchovávanie hodnoty danej lexémy. Túto hodnotu potrebuje nie len generátor kódu, ale je dôležitá aj pri sémantických kontrolách, preto sa posielajú spolu s názvom tokenu parseru.

Atribúty môžu byť rozdielne alebo mať rovnaký typ. Pri rovnakom type má každý typ tokenu svoju tabuľku, do ktorej si ukladá hodnoty atribútov, názov tokenu je teda reprezentovaný číslom tejto tabuľky a atribút poradím jeho hodnoty v tabuľke. Pri ukladní rozdielnych hodnôt je hodnota atribútu priamo doň uložená. Pri vstupnom znaku + zo zdrojového súboru by názov tokenu bol plus a atribút by zostal prázny. [6], [1]

[**int**, *60*] - token s rozdielnymi atribútmi



Obrázok 5.1.6.1. Token s rovnakými atribútmi Prevzaté z [6]

Predpokladajme, že každý token má najviac jeden priradený atribút. V tomto atribúte sa však môže nachádzať viac druhov informácií. Tokeny pre identifikátory sú v tomto prípade najlepším a najdôležitejším príkladom. Zvyčajne sa informácie o lexéme, type či lokácii prvého výskytu ukladajú do tabuľky symbolov. Preto je vhodnejšie atribút tokenu nastaviť na ukazateľ do tabuľky symbolov.

Príklad 5.1.7. Nech vstupný súbor obsahuje nasledujúci údaj:

A = B * C - 8

Z tohto údaju by boli vytvorené nasledujúce tokeny:

<token_identifikátor,ukazateľ do tabuľky symbolov na A>
<token_rovné>
<token_identifikátor,ukazateľ do tabuľky symbolov na B>
<token_násobenie>
<token_identifikátor,ukazateľ do tabuľky symbolov na C>
<token_odčítanie>
<token_celé_číslo, celé číslo s hodnotou 8>

Príklad inšpirovaný [1].

Možno si všimnúť, že tokeny pre operátory majú atribút prázdny. Okrem nich sa atribút neuvádza ani pri interpunkcii či kľučových slovách. V tomto príklade bola do tokenu **celé číslo** priradená priamo hodnota uvedená vo vstupnom súbore. V praxi by sa v kompilátore uložil znakový reťazec reprezentujúci konštantu a použil by sa ako atribút pre token **celé číslo** ukazateľ na tento reťazec[1].

5.1.8 Tabuľka symbolov

Keďže je potreba uchovávať rôzne typy informácií o identifikátoroch, používa sa tabuľka symbolov, ktorá okrem názvu identifikátora obsahuje aj informácie o tom, akého dátového typu je, či je to premenná alebo konštanta a v takom prípade aj jej hodnotu. Identifikátory sa nevzťahujú len na premenné, ale aj na názvy funkcií. O nich sú v tabuľke uložené informácie o počte a typoch jednotlivých parametrov a o ich použití či definícii.

Tabuľku symbolov teda možno definovať ako štruktúru obsahujúcu množinu alebo multimnožinu dvojíc kľúč-hodnota, kde kľúčom obyčajne býva identifikátor a hodnota sú už vyššie spomínané informácie. [3]

Tabuľka symbolov	
Kľúč:	*Informácie*
ID1	Premenná, Typ: integer
Pi	Konštanta, Typ:real ,Hodnota: 3.14159
foo	Funkcia, Typ: char, Počet parametrov: 2

Obrázok 5.1.9. Štruktúra tabuľky symbolov. Prevzaté z [6]

5.2 Rozdiely medzi lexikálnou a syntaktickou analýzou

Ako bolo uvedené v 5.1, lexikálna analýza okrem spracovania lexémov zastáva aj iné úlohy. Odstránenie komentárov, medzier a voľných miest by mohla vykonávať syntaktická analýza, no takýto parser by bol značne zložitejší. Vďaka oddeleniu lexikálnej a syntaktickej časti sa dosiahne čistejší celkový dizajn jazyka. Okrem toho je kompilátor efektívnejší, pretože oddelený lexikálny analyzátor povoľuje aplikáciu špeciálnych techník, ktoré sú špecifické pre lexikálne úlohy, syntaktická analýza ich nepotrebuje. Naviac, práca s vyrovnávacou pamäťou pri čítaní vstupných znakov značne urýchľuje kompilátor. Taktiež práca so samotným vstupným súborom je vyhradená pre lexikálny analyzátor.

5.3 Syntaktická analýza

Druhou fázou prekladu je syntaktická analýza. Jej vstupom je výstup lexikálneho analyzátoru, teda postupnosť tokenov. Úlohou syntaktického analyzátoru (parseru) je zistiť, či táto postupnosť patrí do zdrojového jazyka generovaného gramatikou, čiže či je zdrojový súbor zapísaný správne syntakticky. Ak nie, požadujeme, aby parser chyby oznámil, prípadne sa z nich zotavil. Pokiaľ je zdrojový súbor v poriadku, syntaktický analyzátor vyprodukuje svoj výstup - derivačný strom, ktorý pošle zvyšku kompilátoru na ďalšie spracovanie. Tieto stromy reprezentujú syntaktickú štruktúru zdrojových programov a korešpondujú s ľavými (a pravými) deriváciami, preto je výhodnejšie hľadať pre daný reťazec ľavú (alebo pravú) deriváciu.

Existujú tri obecné typy parserov pre gramatiky: univerzálna, zhora-nadol a zdola-nahor. Univerzálne metódy dokážu spracovať akúkoľvek gramatiku, no ich použitie je neefektívne pri výrobe kompilátorov. Príklady týchto metód sú napríklad algoritmus Cocke-Younger-Kasami alebo Earley-ho algoritmus.

Zvyčajne používané metódy v kompilátoroch sú buď metóda zhora-nadol alebo metóda zdola-nahor. Ako napovedajú ich mená, metóda zhora-nadol stavia derivačný strom z vrchu (koreňa) až ku koncu (listy). Naopak metóda zdola-nahor začína od listov a prepracuje sa nahor do koreňa. V každom prípade je však vstup parsera čítaný po jednom symbole zľava doprava. Tieto dve metódy pracujú len s úzkou skupinou gramatík, niektoré z nich, konkrétne LL a LR gramatiky, sú dostatočne výrazné na to, aby dokázali popísať väčšinu konštrukcie syntax v moderných programovacích jazykoch. Pri použití LL gramatík sa často parser implementuje ručne, LR gramatiky sú zvyčajne konštruované pomocou automatizovaných nástrojov. [1]

5.3.1 LL a LR gramatika

Príklad 5.3.1.1. Rozdiel medzi LL a LR gramatikami. Prevzaté z [1]

$$
\begin{aligned}
&E \to TE' \\
&E' \to +TE'|\epsilon \\
&T \to FT' \\
&T' \to *FT'|\epsilon \\
&F \to (E)|\mathbf{id}
\end{aligned}
$$

$$
\begin{aligned}
&E \to E + T|T \\
&T \to T * F|F \\
&F \to (E)|\mathbf{id}
\end{aligned}
$$

V predošlých dvoch gramatikách si možno všimnúť rozdiely medzi LL a LR gramatikami. Vrchná časť je ľavo-nerekurzívna LL gramatika, ktorá je vhodná pri metóde zhora-nadol. Naopak, spodné tri pravidlá prislúchajú LR gramatike a ich použitie je vhodné pri metóde zdola-nahor. Hoci táto gramatika zvládne pridanie operátorov a ich prioritu, kvôli ľavej rekurzívnosti ju nie je možné použiť pri metóde zhora-nadol.

Konštrukcia parserov prebieha za pomoci funkcii *First* a *Follow*, ktoré prislúchajú ku gramatike G. S ich pomocou sa počas analýzy zhora-dolu vyberajú pravidlá na základe ďalšieho symbolu.

Množina $First(x)$ je množina všetkých terminálov, ktorými môže začínať reťazec derivovateľný z x. [6]

Definícia 5.3.1.2. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Pre každé $x \in (N \cup T)^*$ je definované $First(x) = \{a : a \in T, x \Rightarrow^* ay; y \in (N \cup T)^*\}$

Množina $Empty(x)$ je množina, ktorá obsahuje jediný prvok ϵ, ak x derivuje ϵ, inak je prázdna.[6]

Definícia 5.3.1.3. Nech $G = (N, T, P, S)$ je bezkontextová gramatika.
$Empty(x)= \{\epsilon\}$ ak $x \Rightarrow^* \epsilon$; inak
$Empty(x)= \emptyset$, kde $x \in (N \cup T)^*$.

Množina $Follow(A)$ je množina všetkých terminálov, ktoré sa môžu vyskytovať vpravo od A vo vetnej forme.[6]

Definícia 5.3.1.4. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Pre všetky $A \in N$ definujeme množinu $Follow(A)$:

$$Follow(A) = \{a : a \in T, S \Rightarrow^* xAaz, x, z \in (N \cup T)^*\} \cup \{\$: S \Rightarrow^* xA, x \in (N \cup T)^*\}$$

Množina $Predict(A \to x)$ je množina všetkých terminálov, ktoré môžu byť aktuálne najľavejšie vygenerované, pokiaľ pre ľubovoľnú vetnú formu použijeme pravidlo $A \to x$.[6]

Definícia 5.3.1.5. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Pre každé $A \to x \in P$ definujeme množinu $Predict(A \to x)$ako:

- ak $Empty(x) = \{\epsilon\}$ potom:
$$Predict(A \to x) = First(x) \cup Follow(A)$$
- inak ak $Empty(x) = \emptyset$ potom:
$$Predict(A \to x) = First(x)$$

Množiny $First(x)$,$Empty(x)$ a $Follow(A)$ sa podieľajú na vzniku množiny $Predict(A \to x)$, na základe ktorej sa zostavuje LL tabuľka. Táto tabuľka pomáha pri syntaktickej analýze rozdnúť, ktoré pravidlo bude aplikované na daný neterminál.

5.3.2 Syntaktická analýza zhora-nadol

Ako už bolo uvedené v 5.3, metóda zhora-nadol sa zaoberá konštukciou derivačného stromu pre vstupný reťazec začínajúc od koreňa k listom. Taktiež môžme za cieľ tejto metódy považovať nájdenie najľavejšej derivácie vstupného reťazca.[1]

Definícia 5.3.2.1. Nech $G = (N, T, P, S)$ je bezkontexotvá gramatika, ktorá má n pravidiel očíslovaných $1, ..., n$ a nech w je z jazyka generovného touto gramatikou.
Syntaktická analýza zhora-nadol je proces, ktorého použitím sa nájde postupnosť čísel pravidiel použitých pri ľavej derivácii vety w. [7]

Pri syntaktickej analýze zhora-nadol je kľúčovým problémom zistenie pravidla, ktoré bude aplikované na aktuálny neterminál. Keď sa takéto pravidlo nájde, zbytok procesu syntaktickej analýzy pozostáva z porovnávania vstupných symbolov s telom pravidla.
Rekurzívny zostup je názov obecnej formy metódy zhora-nadol. Môže však vyžadovať spätné trasovanie (*backtracking*), aby vhodné pravidlo našla.

Rekurzívny zostup

Rekurzívny zostup pozostáva z množiny procedúr, jednej pre každý neterminál. Vykonanie tejto metódy začína od počiatočného symbolu, ktorý hlási úspech, ak telo jeho procedúry oskenuje celý vstupný reťazec.[1]

Algoritmus 2: Rekurzívny zostup [1]

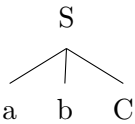
```
Vyber pravidlo, A → X_1 X_2 ... X_k;
for i = 1 po k do
    if X_i je neterminál then
        volanie procedúry X_i();
    end
    else if X_i sa rovná aktuálnemu vstupnému symbolu a then
        pokračovať na ďalší symbol na vstupe;
    end
    else
        /*nastal problém*/
    end
end
```

Obecne môže rekurzívny zostup vyžadovať opakované skenovanie vstupu, teda spätné trasovanie. Tento postup je však potrebný len ojedinele. Spätné trasovanie nie je veľmi efektívne, a to ani pri spracovávaní prirozdených jazykov.[1]

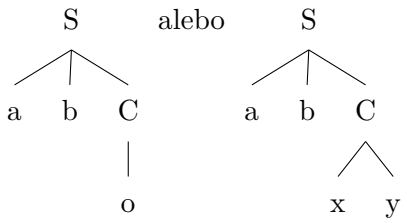
Príklad 5.3.2.2. Majme gramatiku:

$$S \to abC$$
$$C \to o|xy$$

Na vstupe máme reťazec $r = abxy$. Pri konštrukcii derivačného stromu začneme koreňovým uzlom, ktorý označíme S. Ukazateľ na vstup bude ukazovať na prvý symbol reťazca r, a. Keďže symbolom S začína iba jedno pravidlo, použijeme práve toto na expanziu S a získame nasledujúci strom:



Najľavejší list a sa zhoduje s prvým znakom vstupného reťazca, takže sa ukazateľ na vstup posunie na druhý znak r, a porovnávať budeme s ďalším listom, b. Znova bola nájdená zhoda. Ukazateľ na vstup znova posunieme, tentoraz na x. Ďalším listom, ktorý potrebuje expanziu, je C. Ponúkajú sa dve možnosti, teda derivačný strom by mohol vyzerať nasledovnými spôsobmi:



Po expanzii sme sa dostali k dvom alternatívam. V prvej z nich by sme ukazateľ na vstup porovnávali s listom o. Tieto znaky sa však nezhodujú, preto oznámime chybu a vrátime sa do C a použijeme druhú alternatívu. List x sa zhoduje z tretím symbolom reťazca r, takže ukazateľ na vstup opäť presunieme a porovnávať budeme s listom y. Znova sme našli zhodu a teda prešli celý vstupný reťazec r. Keďže bol vygenerovaný celý derivačný strom pre r, ukončíme proces a ohlásime úspech derivácie. Príklad inšpiriovaný [1].

Prediktívna syntaktická analýza

Prediktívna analýza patrí medzi formu analýzy zhora-nadol, na rozdiel od rekurzívneho zostupu však vyžaduje explicitnú implementáciu zásobníku. Pri zmene gramatiky nutno zmeniť iba tabuľku, nie celú implementáciu ako pri rekurzívnom zostupe.
Nasledujúci symbol jednoznačne rozhoduje, ktorým smerom sa bude uberať kontrola pre každý neterminál. Sekvencia volaní procedúr počas analýzy vystupného reťazca implicitne definuje derivačný strom na základe vstupu, a môže byť použitý na stavbu explicitného derivačného stromu. Prediktívna analýza môže byť konštruovaná pomocou triedy LL(1) gramatík. Parser pre prediktívnu analýzu obsahuje procedúru pre každý neterminál. [1]

Algoritmus 3: Prediktívna analýza [6]

push(\$) a push(S) na zásobník
while *úspech alebo chyba* **do**
  Nech X je vrchol zásobníku a a je aktuálny token
  switch X **do**
    case $X = \$$ **do**
    if $a = \$$ **then**
      úspech
    end
    else
      chyba
    end
    ;
    case $X \in T$ **do**
    if $X = a$ **then**
      pop(X) a prečítaj ďalšie a zo vstupného reťazca
    end
    else
      chyba
    end
    ;
    case $X \in N$ **do**
    if $r : X \Rightarrow x \in LL - tabulka[X, a]$ **then**
      zameň na vrchole zásobníka X za $reversal(x)$ a zapíš r na výstup
    end
    else
      chyba
    end
    ;
  end
end

5.3.3 Syntaktická analýza zdola-nahor

Syntaktická analýza zdola-nahor vytvára derivačný strom pre vstupný reťazecc z listov ku koreňu. Obecný spôsob analýzy zdola-nahor sa nazýva precedenčný syntaktický analyzátor. Okrem neho sa používa aj LR syntaktický analyzátor, a hoci je jeho implementácia náročnejšia, je silnejší. No práve kvôli jeho zložitosti sa nezvykne implementovať ručne ale s pomocou automatických generátorov parserov. Potrebné je mať zostavenú vhodnú LR gramatiku. O syntaktickej analýze zdola-nahor môžme povedať, že je to proces „redukovania“ reťazca w na poriačočný symbol gramatiky. Počas každého redukčného kroku sa porovnáva podreťazec tela pravidla, a ak sa zhoduje, je nahradený neterminálom na začiatku pravidla. Najdôležitejšie rozhodovanie počas tohto typu analýzy je *kedy* redukovať a *aké* pravidlo použiť. [1], [6]

Definícia 5.3.3.1. Nech $G = (N, T, P, S)$ je bezkontexotvá gramatika, ktorá ma n pravidiel očíslovaných $1, ..., n$ a nech w je z jazyka generovného touto gramatikou.
Syntaktická analýza zdola-nahor je proces, ktorého použitím sa nájde obrátená po-

stupnosť čísel pravidiel použitých pri pravej derivácii vety w.

Precedenčná analýza

Precedenčná analýza, alebo *shift-reduce parsing*, je forma analýzy zdola-nahor. Používa zásobník, na ktorý si ukladá symboly gramatiky. Jeho spodok značí $, čo je taktiež znak konca výrazu na zanalyzovanie. Vstup je skenovaný zľava doprava a parser presúva na zásobník žiaden alebo viac vstupných symbolov a vytvára tak reťazec znakov z gramatiky, β. Keď je na zásobníku počet znakov taký, že je možná ich redukcia, vykoná sa na základe vhodného pravidla. Tento cyklus je opakovaný dovtedy, kým nenarazí na chybu alebo nie je spracovaný celý vstupný výraz. [1]

Algoritmus 4: Precedenčná analýza [6]

```
Nech funkcia top vracia terminál na zásobníku najbližšie vrcholu
Vloženie $ na zásobník
while b ≠ $ alebo zásobník neobsahuje len S do
    Nech a = top
    Nech b = aktuálny znak na vstupe
    switch Tabuľka[a, b] do
        case = do push(b) & prečítaj ďalší symbol b zo vstupu
        ;
        case < do Zameň a za a < na zásobníku & push(b) & prečítaj ďalší
         symbol b zo vstupu ;
        case > do
        if < y je na vrchole zásobníku and r : A → y ∈ P then
         | Zameň < y za A a zapíš r na výstup
        end
        else
         | chyba
        end
        ;
        case prázdne políčko do Chyba ;
    end
end
```

Precedenčný parser môže vykonávať štyri akcie:[1]

- **Shift** - vloženie ďalšieho vstupného symbolu na zásobník
- **Reduce** - pravý koniec reťazca na redukovanie musí byť na vrchole zásobníku. Nájde sa ľavý koniec tohto reťazca na zásobníku a rozhodne sa, ktorým neterminálom bude reťazec nahradený.
- **Accept** - úspešné dokončenie analýzy
- **Error** - syntaktická chyba bola objavená a treba ju nahlásiť

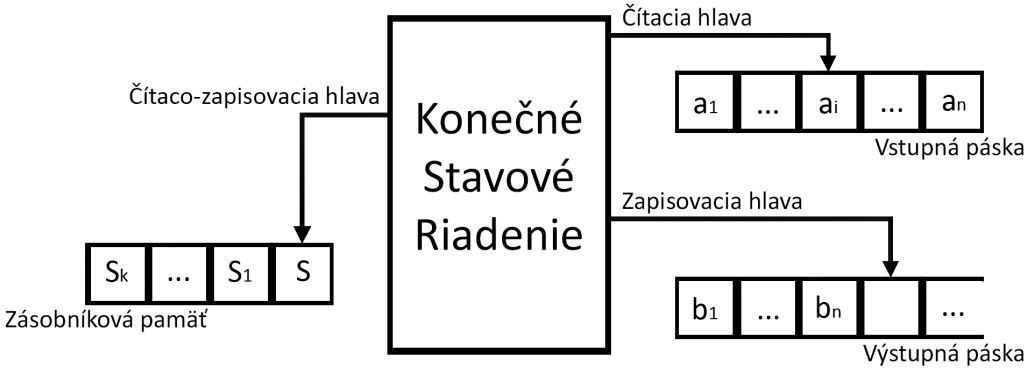
Zhrnutie

Lexikálna a syntaktická analýza sú prvé dve fázy pri preklade zdrojového súboru do cieľového programu. V tejto časti práce sme priblížili ich význam z hľadiska prekladu, pri lexikálnej analýze sme predstavili token, čo je výstup lexikálneho analyzátoru skladajúci sa z dvoch položiek - typu a atribútu. Vysvetlili sme si, aké hodnoty tieto položky môžu obsahovať a na obrázku sme ilustrovali vytvorenie tokenu. Syntaktická analýza, druhá časť kompilátoru, používa parser na zistenie, či je zdrojový súbor zapísaný syntakticky správne. Bežne sa používajú dve metódy: zhora-nadol a zdola-nahor, kde prvá z nich pracuje s LL gramatikou a druhá s LR gramatikou. Obe metódy boli v tejto kapitole popísané, konkrétne sme sa zamerali na rekurzívny zostup, preditívnu analýzu, ktoré spadajú pod metódu zhora-nadol, a precedenčnú analýzu, metódu analýzy zdola-nahor.

Kapitola 6

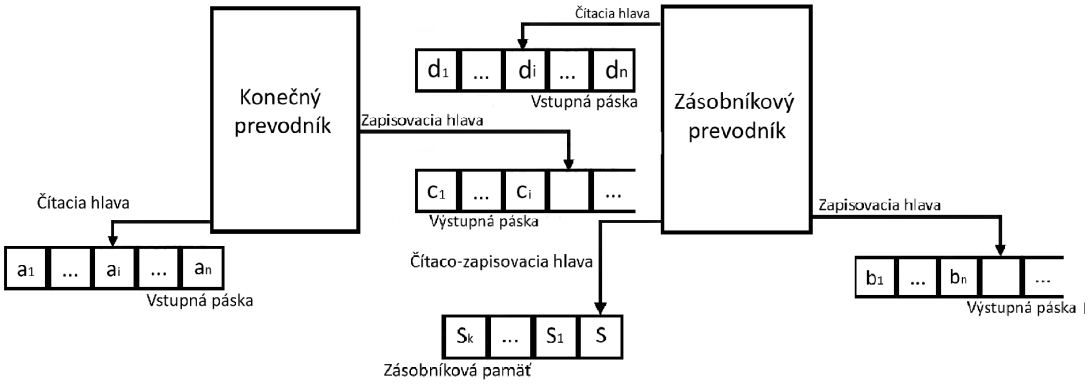
Lexikálno-syntaktický prevodník

Lexikálno-syntaktický prevodník M je konštrukcia zložená z dvoch častí - konečného prevodníku zastupujúceho lexikálnu analýzu a zásobníkového prevodníku predstavujúceho syntaktickú analýzu. Vzhľadom na tieto konštrukcie je lexikálno-syntaktický prevodník (skrátene LST prevodník) zložený zo vstupnej a výstupnej pásky pre konečný prevodník a zo vstupnej a výstupnej pásky a zásobníkovej pamäte pre zásobníkový prevodník. Vstupná páska konečného prevodníku predstavuje celkový vstup LST prevodníku, výstupná páska zásobníkového prevodníku predstavuje celkový výstup LST prevodníku a výstup konečného prevodníku sa stane vstupom pre zásobníkový prevodník. Vstupné a výstupné pásky možno rozdeliť na štvorce, po ktorých sa čítacia a zapisovacia hlava budú pohybovať. Ak vstupná páska LST (konečného) prevodníku obsahuje reťazec $a_1...a_i...a_n$ a čítacia hlava sa nachádza nad štvorcom obsahujúcim symbol a_i, tento symbol je aktuálnym vstupným symbolom. Výstupná páska LST (zásobníkového) prevodníka sa doprava nekonečne predlžuje, môžeme o nej teda povedať, že je polo-nekonečná. Ak obsahuje reťazec $b_1...b_n$, zapisovacia hlava sa nachádza nad prázdnym štvorcom za štvorcom obsahujúcim symbol b_n. Zásobníková páska ovplyvňuje každý prechod vykonaný LST prevodníkom. Prvý symbol zásobníkovej pamäte sa nachádza najľavejšie.



Obrázok 6.1. Celkový pohľad na LST prevodník

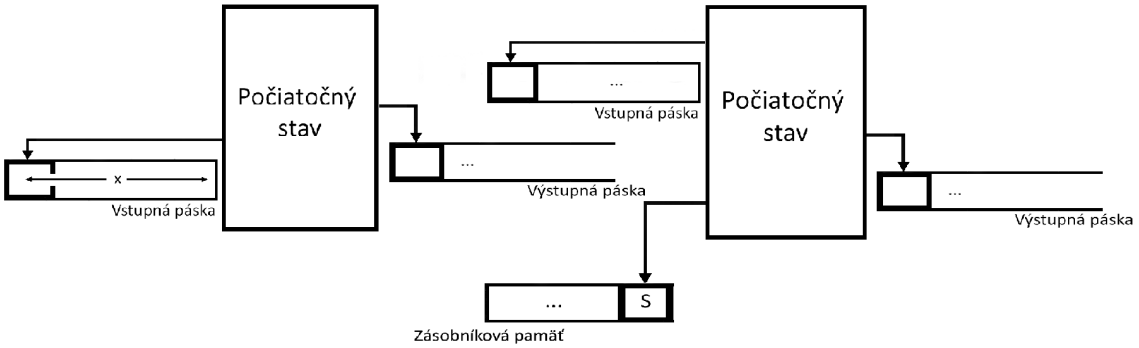
Z obrázku si možno všimnúť, že vonkajšia konštrukcia LST prevodníka sa podobá štruktúre zásobníkového prevodníku. LST prevodník je však zložený z oboch - konečného aj zásobníkového prevodníku, čo ilustruje nasledujúci obrázok.

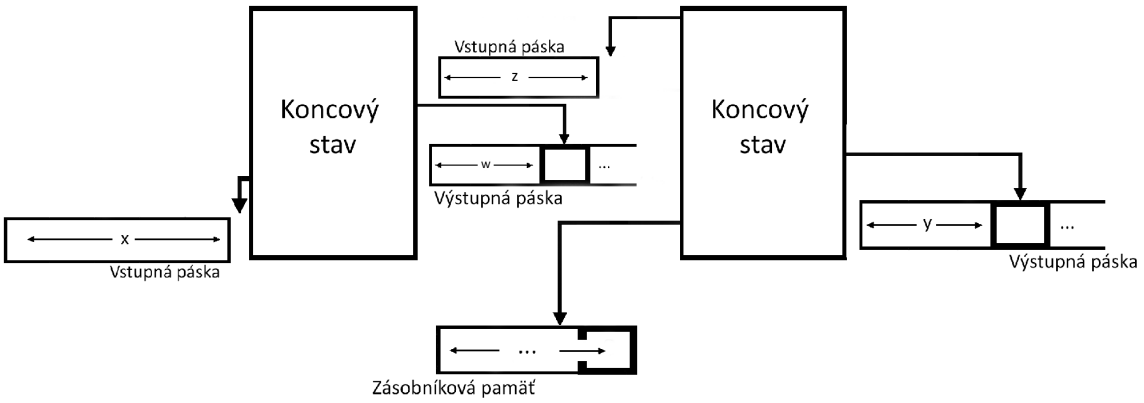


Obrázok 6.2. Konkrétny pohľad na LST prevodník

Pre vstupnú pásku zásobníkového prevodníku platí to isté ako pre vstupnú pásku konečného prevodníku, ak teda obsahuje reťazec $d_1...d_i...d_n$ a čítacia hlava sa nachádza nad štvorcom obsahujúcim symbol d_i, tento symbol je aktuálnym vstupným symbolom. Ak výstupná páska konečného prevodníku obsahuje reťazec $c_1...c_i...$, tak sa zapisovacia hlava nachádza o jeden štvorec za štvorcom obsahujúcim symbol c_i. Na skopírovanie obsahu výstupnej pásky konečného automatu využíva LST komunikačné symboly.

Východzí stav lexikálno-syntaktického prevodníku je stav, kedy sa na vstupnej páske náchadza reťazec x, na vrchu zásobníkovej pamäte sa nachádza symbol S, vstupná páska konečného prevodníku a výstupné pásky sú prázdne a M sa nachádza v počiatočnom stave. Preklad x na y pomocou Lexikálno-Syntaktického prevodníku môžme nazvať postup, keď M prečíta celý reťazec x zo vstupnej pásky, zapíše reťazec w na výstupnú pásku konečného prevodníka, skopíruje jej obsah na vstupnú pásku zásobníkového prevodníku pomocou komunikačných symbolov, vypíše y na výstupnú pásku, a z počiatočného stavu dosiahne koncový stav.





Obrázok 6.3. Preklad LST prevodníkom

Aparáty LST prevodníku, konečný a zásobníkový prevodník, možno prirovnať ku gramatickým systémom paralelne komunikujúcich gramatických systémov. V kapitole 3 bolo vysvetlené ich fungovanie, rozdelenie na komponenty *master* a *slave* a komunikácia pomocou komunikačných symbolov. Zásobníkový prevodník by sme mohli prirovnať ku komponente *master*, pretože kontroluje deriváciu výstupu LST prevodníku a jeho prechody môžu obsahovať komunikačné symboly, *slave* komponentou by bol konečný prevodník, ktorý odpovedá na žiadosť zásobníkového automatu a predá mu svoj výstup - token. Analogicky ku PCGS, výsledok LST prevodníku je výsledná derivácia zásobníkového prevodníku nehľadiac na to, či má konečný prevodník ešte nejaké symboly na vstupe. V prípade, že je výstup LST prevodníku chyba, a teda na vstupe bol zadaný chybný zdrojový kód, vstup konečného prevodníku nemusí byť prázdny. Naopak, v prípade zdrojového kódu, ktorý chybu neobsahuje, bude vstup oboch aparátov prázdny.

Definícia 6.4. Lexikálno-syntaktický prevodník (LST) je dvojica:
$M = (M_1, M_2)$, kde
M_1 je konečný prekladový automat $M_1 = (Q, \Sigma, R, s, F)$, kde

- Q je konečná množina stavov
- Σ je abeceda taká, že $\Sigma \cap Q = \emptyset$ a $\Sigma = \Sigma_I \cup \Sigma_O$, kde Σ_I je vstupná abeceda a Σ_O je výstupná abeceda
- $R \subseteq Q(\Sigma_I \cup \{\epsilon\}) \times Q\Sigma_O^*$ je konečná relácia
- $s \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

a M_2 je zásobníkový prekladový automat $M_2 = (Q, \Sigma, R, s, F)$, kde

- Q je konečná množina stavov
- Σ je abeceda taká, že $\Sigma \cap Q = \emptyset$ a $\Sigma = \Sigma_I \cup \Sigma_O \cup \Gamma$,kde Σ_I je vstupná abeceda a Σ_O je výstupná abeceda a Γ je zásobníková abeceda obsahujúca počiatočný symbol S
- $R \subseteq \Gamma Q(\Sigma_I \cup \{\epsilon\}) \times \Gamma^* Q\Sigma_O^*$ je konečná relácia
- $s \in Q$ je počiatočný stav

- $F \subseteq Q$ je množina koncových stavov

Potom $(X_1, X_2) \Rightarrow (\bar{X_1}, \bar{X_2})$, kde

- X_1 je konfiguráciou M_1
- X_2 je konfiguráciou M_2

Definícia 6.5. Lexikálno-syntaktický prekladač (LST) je konštrukcia:
$M = (Q, (\Sigma_1, R_1), (\Sigma_2, R_2), K, s, F)$, kde

- Q je konečná množina stavov
- Σ_1 je abeceda taká, že $\Sigma_1 \cap Q = \emptyset$ a $\Sigma_1 = \Sigma_{1_I} \cup \Sigma_{1_O}$, kde Σ_{1_I} je vstupná abeceda a Σ_{1_O} je výstupná abeceda
- Σ_2 je abeceda taká, že $\Sigma_2 \cap Q = \emptyset$ a $\Sigma_2 = \Sigma_{2_I} \cup \Sigma_{2_O} \cup \Gamma$,kde Σ_{2_I} je vstupná abeceda a Σ_{2_O} je výstupná abeceda a Γ je zásobníková abeceda obsahujúca počiatočný symbol S
- $R1 \subseteq Q(\Sigma_{1_I} \cup \{\epsilon\}) \times Q\Sigma_{1_O}^*$ je konečná relácia
- $R2 \subseteq \Gamma Q(\Sigma_{2_I} \cup \{\epsilon\}) \times \Gamma^* Q\Sigma_{2_O}^*$ je konečná relácia
- K je konečná množina komunikačných symbolov $K = \{Q_1, ..., Q_n\}$
- $s \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

Definícia 6.6. Nech $M = (Q, (\Sigma_1, R_1), (\Sigma_2, R_2), K, s, F)$ je lexikálno-syntaktický prevodník. Dvojicou $(p, a), p \in Q, x \in \Sigma_1^*$ a trojicou $X_2 = (q, w, z)$, kde $q \in Q, w \in \Sigma_2^*$ a $z \in \Gamma^*$ nazývame **konfiguráciu** lexikálno-syntaktického prekladača.

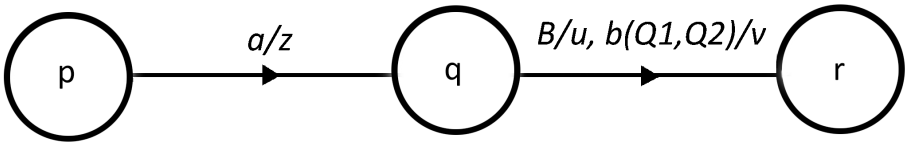
Nech pa, qz, $Bqb(Q_1, Q_2)$ a urv sú konfigurácie lexikálno-syntaktického prekladača M, kde

- $p, q, r \in Q$
- $a \in \Sigma_{1_I} \cup \{\epsilon\}$
- $z \in \Sigma_{1_O}^*$
- $B \in \Gamma$
- $b \in \Sigma_{2_I}$
- $Q_1, Q_2 \in K$
- $u \in \Gamma^*$
- $v \in \Sigma_{2_O}^*$

Definícia 6.7. Výpočtový krok
Nech $o = pa \Rightarrow qz \in R_1$ je pravidlo a $m = Bqb(Q_1, Q_2) \Rightarrow urv \in R_2$ je pravidlo. Potom M môže previesť prechod z pa do qz a $Bqb(Q_1, Q_2)$ do urv zapísané:

- $pa \Rightarrow qz[o]$
- alebo zjednodušene $pa \Rightarrow qz$
- a $Bqb(Q_1, Q_2) \Rightarrow urv[m]$
- zjednodušene $Bqb(Q_1, Q_2) \Rightarrow urv$

Pozn. Ak $a = \epsilon$, zo vstupnej pásky sa symbol neprečíta



Obrázok 6.8. Zobrazenie výpočtového kroku

Znamená to teda, že prevodník najprv prejde zo stavu p do stavu q, pričom zo vstupnej pásky prečíta symbol a a na výstupnú pásku konečného prevodníku zapíše symbol z, následne zo stavu q prejde do stavu r a symbol B na vrchu zásobníkovej pamäte bude nahradený reťazcom u, zo vstupne pásky prečíta symbol b a pomocou komunikačných symbolov z výstupnej pásky konečného prevodníka skopíruje znak z a na výstupnú pásku zapíše symbol v.

Príklad 6.9. Nech existuje Lexikálno-syntaktický prevodník $M = (Q, (\Sigma_1, R_1), (\Sigma_2, R_2), K, s, F)$, kde:

- $Q = \{p, q, r\}$
- $\Sigma_{1_I} = \{a, b\}$
- $\Sigma_{1_O} = \{x, y, z\}$
- $\Sigma_{2_I} = \{x, y, z\}$
- $\Sigma_{2_O} = \{1, 2, 3, 4\}$
- $\Gamma = \{S, O\}$
- $R_1 = \{\mathbf{p}a \Rightarrow \mathbf{q}z, \mathbf{p}b \Rightarrow \mathbf{q}z, \mathbf{q}a \Rightarrow \mathbf{q}x, \mathbf{q}b \Rightarrow \mathbf{q}y, \mathbf{q}\epsilon \Rightarrow \mathbf{r}\epsilon\}$
- $R_2 = \{S\mathbf{q}\epsilon(Q1, Q2) \Rightarrow S\mathbf{q}\epsilon, S\mathbf{q}x(Q1, Q2) \Rightarrow \mathbf{q}1, S\mathbf{q}y(Q1, Q2) \Rightarrow S\mathbf{q}2, S\mathbf{q}z(Q1, Q2) \Rightarrow S\mathbf{q}3, \epsilon\mathbf{r}x \Rightarrow \epsilon\mathbf{r}4, \epsilon\mathbf{r}y \Rightarrow \epsilon\mathbf{r}4, \epsilon\mathbf{r}z \Rightarrow \epsilon\mathbf{r}4\}$
- $K = \{Q_1, Q_2\}$
- $s = p$

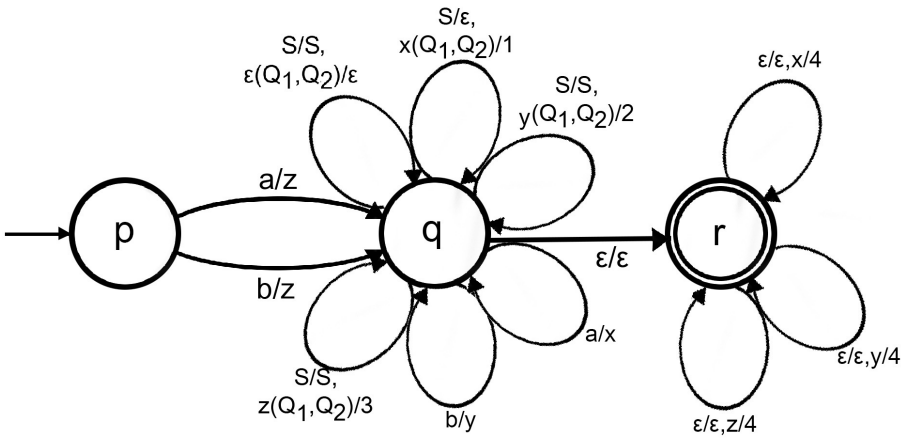
- $F = \{r\}$

Ak bude vstupný reťazec *abab*, LST prevodník vykoná nasledujúcu postupnosť krokov:

$$\mathbf{p}abab \Rightarrow \mathbf{q}z \text{ a } S\mathbf{q}\epsilon(Q1, Q2) \Rightarrow S\mathbf{q}\epsilon$$
$$\mathbf{q}bab \Rightarrow \mathbf{q}y \text{ a } S\mathbf{q}z(Q1, Q2) \Rightarrow S\mathbf{q}3$$
$$\mathbf{q}ab \Rightarrow \mathbf{q}x \text{ a } S\mathbf{q}y(Q1, Q2) \Rightarrow S\mathbf{q}32$$
$$\mathbf{q}b \Rightarrow \mathbf{q}y \text{ a } S\mathbf{q}x(Q1, Q2) \Rightarrow \epsilon\mathbf{q}321$$
$$\mathbf{q}\epsilon \Rightarrow \mathbf{r}\epsilon \text{ a } \epsilon\mathbf{r}y \Rightarrow \epsilon\mathbf{r}3214$$

Reťazec *abab* bol konštrukciou LST preložený na výsledný reťazec 3214. Ak by sme pre preklad použili konečný a zásobníkový prevodník samostane, najpr by sa reťazec preložil na *zyxy* a až potom na výsledný reťazec. V LST prevodníku však komponenty pracujú paralelne, preto bol schopný v jednom kroku priamo preložiť symbol a na symbol 4.

LST prevodník z príkladu 6.9 je znázorenní ako automat pozostávajúci z troch stavov (p,q,r), ktorých prechody medzi sebou sú mixom prechodov konečného a zásobníkového prevodníku, čo dokazuje, že oba aparáty pracujú paralelne. Pravidlá z R_1 patria ku konečnému prevodníku, pravidlá z R_2 zas k zásobníkovému. V nich si možno všimnúť komunikačné symboly Q_1 a Q_2, ktoré zabezpečujú skopírovanie hodnoty z výstupu konečného prevodníka na vstup zásobníkového. Keďže činnosť LST prevodníka riadi primárne zásobníkový prevodník, komunikačné symboly sa nachádzajú len v jeho pravidlách.



Obrázok 6.10. Grafické znázornenie LST prevodníka z príkladu 6.9

Zhrnutie

Lexikálno-syntaktický prevodník LST je jednotný aparát popisujúci lexikálnu a syntaktickú analýzu ako jednu jednotku. Skladá sa z konečného a zásobníkového prevodníku, ktoré spolu komunikujú pomocou komunikačných symbolov. Predstavili sme si preklad týmto

prevodníkom, čo bolo doplnené aj graficky a ilustrované na príklade. Okrem definícii sme zaviedli LST prevodník analogicky ku gramatickým systémom k PCGS.

Kapitola 7

Implementácia

V kapitole 3 sme definovali konštrukciu LST zásobníku a všetky jej vlastnosti. Túto konštrukciu je však potrebné aj implementovať, aby sme mohli všetky jej vlastnosti porovnať s už existujúcimi konšturukciami. Táto kapitola obsahuje popis návrhu implemetácie, jej jednotlivých častí a taktiež zavedený jazyk AIDA, ktorý je LST prevodník schopný spracovávať. Implementáciu možno rozdeliť do 3 častí - lexikálnej analýzy, syntakticko-sémantickej analýzy a generovania kódu.

7.1 Konceptuálny návrh

Aplikácia je implementovaná v jazyku C a je spustiteľná na operačnom systéme Linux, prípadne Windows, v takom prípade však musiať byť prístupné podsystémy Linuxu. Aplikácia načíta vstupný súbor obsahujúci vstupný kód v jazyku AIDA a preloží ho do cieľového jazyka AIDAcode, chybové hlásenia a ladiace výpisy sú smerované na štandardný chybový výstup, jedná sa teda o konzolovú aplikáciu. Vstupný kód je založený na jazyku Python 3, pridané sú k nemu však aj prvky z jazyka C. Cieľový jazyk AIDAcode zas zahŕňa trojadresné a zásobníkové inštrukcie, kde sa každá inštrukcia skladá z jej názvu a prípadne operandov. Pri názvoch inštrukcií nezáleží na veľkosti písmen, naopak pri názvoch operandov áno. Návrh implementácie bol inšpirovaný a vychádza zo [4].

7.1.1 Štruktúra zavedeného jazyka

Jazyk, ktorý spracováva naimplementovaný LST prevodník, AIDA, je založený na jazyku Python 3 s prvkami z jazyku C, a to napríklad bodkočiarku či svorkové zátvorky. Jazyk je štruktúrovaný, podporuje definície premenných, užívateľských funkcií, riadiace príkazy ako cyklus *for* a *while*, príkaz priradenia, volanie funkcií a spracovávanie výrazov. Hlavné telo programu je neoznačená sekvencia príkazov. Kdekoľvek v nej sa môžu nachádzať definície užívateľských funkcí. Sekvencia hlavného tela programu je ukončená znakom konca zdrojového súboru. Väčšina konštrukcií využívaných v jazyku AIDA sú jednoriadkové, oddelené od ostatných sú znakom bodkočiarka.

7.1.2 Definícia premenných a užívateľských funkcií

Premenné v jazyku AIDA môžu byť lokálne alebo globálne - lokálne premenné sú definované v tele funkcie a ich platnosť je limitovaná na funkciu, v ktorej boli definované. Globálne premenné sú definované v hlavnom tele programu a rozsah ich platnosti je od miesta definície až po koniec zdrojového programu. Definícia premennej prebieha prvým priradením hodnoty do tejto premennej, špecifický príkaz na jej definíciu nie je zavedený. Každá premenná musí byť definová pred jej použitím a jej meno nesmie byť zhodné s názvom akejkoľvek funkcie definovanej v hlavnom tele. Nesplenenie týchto podmienok je vyhodnotené ako sémantická chyba. Globálne premenné však môžu byť prekryté lokálnou premmennou alebo formálnym parametrom funkcie, ak sa jedná o definíciu užívateľskej funkcie. Užívateľské funkcie musia byť, rovnako ako premenné, definované pred ich volaním.

7.2 Spustenie aplikácie

Implementovanú aplikáciu možno spustiť príkazom **./LST** v príkazovom riadku, pokiaľ už prebehol preklad príkazom **make**. Aplikácia má explicitne nastavený vtup na **stdin** (*štandardný vstup*) a výpis produkovaného kódu je mierený na **stdout** (*štandardný výstup*). Užívateľ ich však môže meniť pomocou nasledujúcich prepínačov:

- **-i** *<názov vstupného txt súboru>*
- **-o** *<názov výstupného txt súboru>*

Oba prepínače môžu byť použíté len raz, v prípade ich chybného zadania je užívateľ upozornený. Taktiež je možné použiť prepínač **help**, ktorý užívateľovi napovie, ako používať prepínače a ktoré súbory môžu byť nastavené ako vstup a výstup.

7.3 Štruktúra LST prevodníku

Ešte predtým, než sa popíše implementácia prekladu pomocu LST prevodníku, je vhodné oboznámiť sa so štrukúrou samotného prevodníku, ktorá sa nachádza v súbore **LST.h**. Prvou komponentou tejto štruktúry je komunikačný symbol, do ktorého sa uloží aktuálny token a reťazec s jeho hodnotou. Pomocou tohto symbolu sú naplnené komponenty *actual_token* a *actual_string*. Komponenty *previous_token* a *previous_string* slúžia na uchovanie predošlého tokenu a jeho reťazca. Význam tohto úkonu bude vysvetlený ďalej v 7.6. Okrem nich štruktúra LST obsahuje premennú pre uloženie jeho aktuálneho stavu, počítadlá pre generovanie náveští, lokálnu a globálnu tabuľku symbolov, pomocné premenné pre uloženie aktuálnych položiek tabuľky pre jednoduchšie použitie, komponentu pre uloženie chyby a zopár pomocných booleovských hodnôt. Táto štruktúra je predávaná funkciám počas syntaktickej analýzy.

7.4 Lexikálna analýza

Lexikálna analýza je prvou časťou LST prevodníka. Jej implementácia vychádza z návrhu konečného automatu priloženého v B a nachádza sa v súboroch **scanner.c** a **scanner.h** . Pri lexikálnej analýze sa zo vstupného súboru postupne načítavajú znaky, zoskupujú sa v lexémach a následne je vytvorená postupnosť internej reprezentácie týchto lexémov - postupnosť tokenov, ktorá je neskôr poslaná zásobníkovému prevodníku na syntaktickú analýzu.

Token je implementovaný ako štruktúra obsahujúca typ tokenu (názov) a jeho atribút. Ten obsahuje číselnú hodnotu či reťazec, v prípade niektorých typov tokenov, napríklad aritmetických, je táto hodnota nezadaná. Pri reťazcoch a komentároch je ich dĺžka dopredu neznáma, preto bolo nutné zaviesť dynamický typ reťazca **string_dynamic**, ktorého veľkosť je možné realokovať podľa potreby a taktiež možnosť pracovať s jeho obsahom jednoduchšie. Implementácia dynamického reťazca sa nachádza v súboroch **string_dynamic.c a string_dynamic.h** Prechod celým vstupným súborom po znakoch je implementovaný vo funkcii *get_next_token*, kde v nekonečnom cykle *while* prebieha načítanie aktuálneho znaku zo vstupného súboru a na základe neho sa postupuje po stavoch konečného automatu. V koncových stavoch dochádza k vytvorenie tokenu na základe vlastností aktuálneho lexému. Stavy konečného automatu sú implementované ako prvky konštukrcie *enum*, teda každému stavu prislúcha označenie celým číslom. Práve táto vlastnosť je využitá pri ich výbere a postupom - celočíselné označenia stavov vyhovujú konštukcii *switch*.
Konštrukcia vyzerá nasledovne:

Algoritmus 5: Prechod stavmi v lexikálnej analýzy

```
while 1 do
    switch aktuálny stav do
        case počiatočný stav do  if aktualny_symbol == '+' then
            prechod do stavu PLUS
        end
        else if aktualny_symbol == '-' then
            prechod do stavu MINUS
        end
        ...
        break;
        case stav PLUS do  if aktualny_symbol == '+' then
            prechod do stavu PLUS_PLUS
        end
        else
            názov tokenu nastav na TOKEN_PLUS; zahoď aktuálny znak vráť
             token
        end
        ...
        break;
    end
end
```

7.5 Komunikačné symboly

Po lexikálnej analýze nasleduje syntaktická a sémantická analýza. Obe však potrebujú výstupné tokeny z lexikálnej analýzy na to, aby mohli zdrojový súbor skontrolovať. Na prevod tokenov z výstupu koenčného prevodníku na vstup zásobníkového boli zavedené komunikačné symboly na obraz paralelne komunikujúcich gramatických symbolov, ktorých fungovanie v rámci LST prevodníka bolo bližšie popísané v kapitole 6. Implementované sú ako štruktúra pozostávajúca z tokenu a dynamického reťazca, do ktorých je priamo načítaný výstupný token z lexikálnej analýzy. Načítanie výstupného tokenu a dynamického reťazca a

ich následné uloženie do komunikačného symbolu zabezpečuje funkcia *copy_token()*. Slúži taktiež na predanie tohto komunikačného symbolu zásobníkovému automatu vykonávajúceho syntaktickú a sémantickú analýzu.

Obrázok 7.5.1. Výmena výstupov medzi jednotlivými komponentami

7.6 Syntaktická analýza

Najdôležitejšou časťou LST prevodníka z implementačného hľadiska je práve parser, ktorý riadi chod prevodníku. Na jeho požiadavku sa skopírujú hodnoty z výstupu lexikálneho analyzátoru pomocou komunikačných symbolov a okrem syntaktickej analýzy sa stará aj o sémantické kontroly a spúšťanie funkcii generátoru. Z hľadiska kontrolovania syntax používa dva postupy - metódou rekurzívneho zostupu a pravidiel LL gramatiky, priloženej v A, kontroluje riadiace príkazy, predecedenčnú analýzu využíva pri overovaní výrazov. Syntaktická analýza rekurzívnym zostupom je implementovaná v súboroch **parser.c** a **parser.h**, analýza výrazov sa nachádza v súboroch **expression.c** a **expression.h**. Overovanie sémantiky rovnako ako aj volanie funkcií generátoru kódu je zastúpené v oboch funkciách.

7.6.1 Rekurzívny zostup

Kapitola 5.3 sa zamerala na LL gramatiky a syntaktickú analýzu pomocou rekurzívneho zostupu, ktoré boli v tejto časti implementácie uplatnené. Najskôr bola vytvorená LL gramatika v prílohe A, na základe ktorej boli implementované funkcie pre jednotlivé neterminály. Príkladom takýchto funkcií sú *program(), param_group(), param(), command, command_end(), a tak ďalej*. Funkcie nesú názov neterminálu, ktorého pravidlá implementujú. Na základe týchto pravidiel kontrolujú, či poradie tokenov odpovedá syntaktickej štruktúre zdrojového jazyka. Token, ktorý porovnávajú, je získaný z komunikačného symbolu vďaka funkcii *set_next_token()*. Pre samotné porovnanie tokenu s požadovaným vstupom je využívaná funkcia *token_look_ahead()*. V prípadoch, kedy dochádza k priradeniu do premennej sa používa funkcia *set_prev_token()*, ktorá aktuálny token a jeho hodnotu uloží do pomocných premenných štruktúry LST prevodníka. Tento krok je dôležitý v prípade načítavania parametrov, pretože ich počet musí byť uložený do položky v tabuľke symbolov 7.6.2 vytvorenej pre konkrétnu funkciu. Pri priradeniach do premenných je tento postup taktiež nutný, keďže aktuálny token je nastavený až na znak = pri overení, či sa aktuálna premenná už nachádza v tabuľke symbolov.

7.6.2 Tabuľka symbolov

Na implementáciu tabuľky symbolov bola použitá hashovacia tabuľka. Ako bolo vysvetlené v kapitole 5.3, tabuľka symbolov sa používa pri lexikálnej aj syntaktickej analýze, v našom prípade primárne na sémantické kontroly. V prípade implementácie LST prevodníka bola použitá len pri tej neskôr zmienenej. Do tabuľky symbolov sa ukladajú identifikátory premenných a funkcií. Položka nachádzajúca sa v tabuľke symbolov obsahuje nasledujúce atribúty:

- **id** - názov identifikátoru a súčasne kľúč na vyhľadávanie v tabuľke
- **Func_Type** - návratový dátový typ pri funkciách
- **Item_Type** - typ identifikátoru (premenná, funkcia)
- **Item_State** - stav identifikátoru (definovaný, nedefinovaný)
- **param_count** - počet parametrov pri funkciách

7.6.3 Analýza výrazov

Postup spracovania výrazov je implementovaný na základe precedenčnej tabuľky, ktorá určuje prioritu operátorov. Táto tabuľka je implementovaná ako matica obsahujúca štyri hodnoty: *Shift, Reduce, Accept a Error*. Indexy do tejto tabuľky slúžia na získanie jednej z týchto hodnôt. Prvý index pochádza z aktuálneho tokenu, z ktorého sa pomocou funkcie *token_name_to_symbol()* získa symbol. Tieto symboly sú reprezentované štruktúrou *Prec_table_index* a každý z jej prvkov obsahuje celočíselnú hodnotu, preto je ich použitie ako indexy vhodné.
Druhý index je získaný z vrcholu zásobníka symbolov. Na základe týchto dvoch prvkov sa rozdohne, či aktuálny symbol získaný z tokenu bude vložený na zásobník (Shift), získa sa redukčné pravidlo a vyberie sa daný počet prvkov zo zásobníka (Reduce), nastala chyba (Error) alebo je výstup akceptovateľný a ukončujeme precedenčnú analýzu (Accept). V prípade, že sa na vrchole zásobníka nachádza identifikátor, celé číslo, reťazec, booleovská hodnota alebo desatinné číslo, je daný symbol zmenený na sybol *NON_TERMINAL* pre jednoduchšie určovanie pravidiel pre redukciu.

7.6.4 Generovanie kódu

Generovanie kódu je poslednou časťou prekladu v LST prevodníku. Volanie funkcií generátoru má na starosti parser pri rekurzívnom zostupe aj pri precedenčnom spracovávaní výrazov. Generátor má okrem generovania trojadresných inštrukcií na starosti aj sémantické kontroly za behu programu. Správe priradené typy, delenie nulou či pretypovanie sú úlohy, ktoré musí zastávať.

7.6.5 Návratové kódy

Po prečítaní znakov celého vstupného súboru a jeho spracovaní program posiela nasledovné návratové kódy:

0: Preklad prebehol v poriadku, na výstup bude vygenerovaný výstupný kód

1: Chyba pri lexikálnej analýze

2: Chyba pri syntaktickej analýze

3: Chyba pri sémantickej analýze

5: Chyba pri zadaní parametrov vo vstupnom súbore

99: Vnútorná chyba

Výstupné kódy sú inšpirované zadaním [4].

Zhrnutie

V tejto kapitole bola popísaná implementácia výslednej aplikácie. Okrem konceptuálneho návrhu aplikácie boli predstavené jazyk AIDA a AIDAcode, kde prvý z nich je vstupný kód založený na jazyku Python 3 s prvkami C a druhý medzikód. Okrem štruktúry zavedeného jazyka bola popísaná aj štruktúra samotného LST prevodníka, implementácia komunikačných symbolov, či tabuľky symbolov. Nezabudlo sa ani na predstavenie prepínačov, ktoré pri spúšťaní aplikáciu poskytujú pomoc, alebo užívateľovi umožňujú zmeniť vstupný a výstupný súbor.

Kapitola 8

Záver

Cieľom tejto práce bolo predstaviť prekladové automaty a zaviesť systém prekladových automatov analogicky ako gramatické systémy, študovať vlastnosti týchto systémov a implementovať formalizácie rôznych častí prevodníkov.

Keďže je text náučného charakteru, neprestajne sa v ňom vyskytujú odborné termíny. Najčastejšie z nich, ako abeceda, reťazec, jeho sufixy, prefixy, reverzácia a podreťazec, jazyk a bezkontextová gramatika, boli pripomenuté v prvej kapitole.

To, že sú gramatiky zaradené do kapitoly k najčastejšie vyskytovaným pojmom iba dokazuje, akú významnú úlohu zastávujú z pohľadu teórie formálnych jazykov. Rozšírenie ich konceptu pomohlo vzniku paralelne komunikujúcich gramatických systémov, označovaných PCGS, ktorých definície a vlastnosti boli uvedené v ďalšej kapitole. Vysvetlená bola komunikácia medzi gramatikami, ktoré systém gramatík tvoria, rozdiely medzi *master* a *slave* gramatikami a príklad ukázal princíp používania komunikačných symbolov medzi nimi.

Rozšírenia sa nedočkali len gramatiky, ale aj automaty. Konkrétne ku konečnému a zásobníkovému automatu bola pridaná výstupná páska aby mohli produkovať výstup. Vďaka tomuto úkonu vznikli konečný a zásobníkový prevodník, ktoré sme si priblížili v nasledujúcej časti tejto práce. Vysvetlili sme ich význam, pohyb čítacej a zapisovacej hlavy pri čítaní zo vstupnej pásky a zapisovaní na výstupnú, čo sme podporili aj grafickým znázornením pre lepšie pochopenie. Okrem toho bola kapitola doplnená definíciami oboch aparátov, ich výpočtového kroku a konfigurácie. Naznačené boli aj rozdiely medzi prevodníkmi a automatmi, čo bolo opäť podporené obrázkami. Princíp prekladu pomocou týchto aparátov bol ilustrovaný na jednoduchých príkladoch.

Konečný a zásobníkový prevodník sa používajú pri preklade, a to konkrétne pri lexikálnej a syntaktickej analýze. Keďže tieto fázy majú význam aj z hľadiska LST prevodníku, venovaná im bola jedna kapitola. Úvodom bol predstavený kompilátor a jeho fázy, pričom sa zvyšok kapitoly zameral na prvé dve z nich. Pri lexikálnej analýze sme uviedli jej základnú charakteristiku a interakciu medzi ňou a syntaktickou analýzou, čo sme ilustrovali aj graficky.

Vstup lexikálnej analýzy je zdrojový program. Prjímanie jeho znakov je teda zásadné pre lexikálnu analýzu a keďže je každý typ lexémy špecifický, vysvetlili sme odlišnosti v analyzovaní pre niektoré z nich. Toto porovnanie vyvrcholilo predstavením deterministického konečného automatu, ktorý je schopný prijímať všetky tieto vstupné symboly v jednom automate.

Záverom tejto časti bola ukážka zobrazujúca implementačnú metódu pre určenie typu lexémy, pretože toto je informácia, ktorú obsahuje výstup lexikálnej analýzy - token. Token sa skladá z názvu (typu) a atribúty. V časti, ktorá sa tokenu priamo venuje, sme opísali,

aké typy hodnôt môžu tieto dve položky obsahovať a demonštrovali sme ich na príklade.

Rovnako ako token, aj tabuľku symbolov môžu používať lexikálna aj syntaktická analýza, v prvej z nich sa však nutne nemusí vyskytovať. Jej prínos v tejto časti analýzy však existuje a preto sme si ho krátko predstavili. Okrem neho sme si pomocou obrázku predstavili aj jej štruktúru.

Lexikálna aj syntaktická analýza pracujú s lexémami zdrojového súboru a sú funkcie, ktoré by mohli vykonávať jedna aj druhá. No rozdelenie ich práce má význam z hľadiska efektivity aj implementácie, čo sme sa dozvedeli v podkapitole venovaj rozdielu medzi nimi.

A keďže už bola popísaná lexikálna analýza, ďalej sme predstavili syntaktickú analýzu. Rovnako ako pri lexikálnej analýze, aj táto fáza prekladu bola najskôr predstavená a spomenul sa jej význam. Ako už bolo spomenuté, gramatiky sú dôležité pri popisovaní jazyka a potrebné sú aj pri syntaktickej analýze. LL a LR gramatiky sa využívajú pri analýze zhora-nadol a zdola-nahor v uvedenom poradí. Z týchto dôvodov boli predstavené a na príklade sme demonštrovali rozdiel medzi nimi. Pri práci s LL a LR gramatikou je dôležité poznať definície funkcií *First, Follow, Empty a Predict*, ktoré sa používajú pri konštrukcii napríklad LL tabuľky pre analýzu zhoda-nadol, preto im bola pridelená jedna podsekcia.

V predošlom odseku sme spomenuli existenciu dvoch typov syntaktickej analýzy - zhora-nadol a zdola-nahor. Každá pracuje na inom princípe, preto sme si ich predstavili jednotlivo. Pri analýze zhora-nadol sme si najprv vysvetlili princíp jej práce a predstavili sme si rekurzívny zostup a prediktívnu syntaktickú analýzu, metódy analýzy zhora-nadol. Prediktívna analýza na rozdiel od rekurzívneho zostupu vyžaduje implementáciu zásobníku. V časti, ktorá sa venuje týmto dvom metódam sme ich porovnali, pričom rekurzívny zostup bol vysvetlený na príklade a prešli sme aj kúsok obecného kódu pre jeho implementáciu.

Analýza zdola-nahor pracuje opačne, teda od listov ku koreňu. V podkapitole, ktorá sa venovala tejto metóde, sme si ukázali jej implementáciu na kúsku kódu a popísali sme akcie *Shift, Reduce, Accept a Error*, ktoré precedenčná analýza môže vykonávať, čím sme uzavreli syntaktickú analýzu a tým aj kapitolu zameriavajúcu sa na prvé dve fázy prekladu.

Keďže sme predstavili všetky komponenty potrebné na zavednie systému prevodníkov analogicky ako gramatické systémy, nasledujúca kapitola zaviedla lexikálno-syntaktický prevodník LST, zložený z oboch, konečného aj zásobníkového prevodníku, medzi ktorými prebieha komunikácia pomocou komunikačných symbolov vytvorených na základe PCGS. Vysvetlené boli jeho vlastnosti, pohyb čítacej, čítaco-zapisovacej a zapisovacej hlavy počas prekladu týmto prevodníkom, jeho základná štruktúra, ktorá bola podložená grafickými znázorneniami LST prevondíku z celkového hľadiska, aj z konkrétneho. Graficky ilustrovaný bol taktiež proces prekladu týmto prevodníkom. Keďže komunikácia medzi aparátmi LST prevodníka bola inšpirovaná gramatickými systémami PC, nasledoval krátky odsek venujúci sa porovnaniu LST prevodníku s týmto systémom gramatík, pričom sa zameralo hlavne na podobnosť komponent jednotlivých systémov. Táto kapitola bola ukončená príkladom, ktorý demonštroval preklad prevodníkom LST, pričom sa taktiež zameral na dokázanie paralelizmu jednotlivých komponent tohto prevodníka.

Po zavedení lexikálno-syntaktického prevodníku prišla na rad jeho implementácia, čomu sa venovala posledná kapitola. Najskôr sme si prešli konceptuálny návrh aplikácie, ktorý sa zameriaval na popis jazykov AIDA, ktorý je založený na jazyku Python3 s prvkami jazyka C, ako napríklad svorkové zátvorky či bodkočiarka na konci príkazov, rovnako ako aj na popis cieľového jazyka AIDAcode, ktorý zahŕňa trojadresné a zásobníkové funkcie. Z jazyka AIDA boli definície premenných a užívateľských funkcií priblížené najviac. Implementovaná bola aj samotná štruktúra LST prevodníku, ktorá bola popísaná v nasledujúcej podkapitole.

Rovnako ako v kompilátore, lexikálna analýza je prvou časťou aj v LST prevodníku a teda spracováva vstupný súbor so zdrojovým kódom. Spomenuli sme nielen súbory, v ktorých sa nachádza jej implementácia, ale taktiež jej princíp. Jej výstupom sú tokeny, ktoré sú implementované ako štruktúra, podobne ako samotný LST prevodník. Keďže niektoré lexémy, napríklad identifikátory, nemajú dopredu stanovenú dĺžku, bolo nutné definovať reťazec, ktorý by bol schopný dynamicky meniť svoju veľkosť. Táto časť práce sa povenovala vysvetleniu jeho fungovania, aj tomu ako je jeho token klasifikovaný.

Prenos tokenov medzi lexikálnou a syntaktickou analýzou bol realizovaný pomocou komunikačných symbolov, jedna podsekcia bola venovaná aj im. Implementované boli ako štruktúra s funkciou, ktorá umožňovala uloženie komunikačného symbolu do štruktúry LST prevodníka, v rámci ktorého bol použitý v syntaktickej analýze.

Syntaktická analýza je najzložitejšia časť prekladu. Používa nielen rekurzívny zostup a LL gramatiku na určovanie správnosti syntax zdrojového súboru, ale aj tabuľku symbolov pri výskyte identifikátorov v kóde. Na spracovanie výrazov používa precedenčnú analýzu. Každej tejto časti bol venovaný úsek textu. Rekurzívny zostup je implementovaný ako skupina funkcií pre každý neterminál vyskytujúci sa na ľavej strane pravidiel v LL gramatike. Spomenul sa aj prenos a porovnanie tokenov.

Tabuľka symbolov bola implementovaná pomocou hashovacej tabuľky, kde položky tejto tabuľky tvorili premenné a funkcie. Kľúčom k jednotlivým položkám bol ich názov. V sekcii, ktorá sa tejto tabuľke venuje, boli spomenuté nie len jej položky, ale aj ich štruktúra.

Spomínaná bola aj precedenčná analýza, ktorá sa starala o spracovanie výrazov. Implementovaná bola na základe precedenčnej tabuľky. Rozobral sa v nej postup pri určovaní redukčných pravidiel, vkladanie na zásobník, či zámena operandov pre jednoduchšiu redukciu symbolom *NON_TERMINAL*.

Generovanie kódu je posledná časť, ktorá bola naimplementovaná. Generátor okrem generovania kódu vykonáva aj sémantické kontroly počas behu programu. Sám však obsahuje len funkcie, tie sú volané parserom.

Výsledkom tejto práce je zavedený LST prevodník, ktorý funguje na podobnom princípe ako paralelne komunikujúce gramatické systémy, pričom využíva konečný a zásobníkový prevodník ako svoje komponenty. Prevodník je schopný vykonávať lexikálnu aj syntaktickú analýzu ako jednu jednotku, pričom nestráca výhody oddelenia lexikálnej a syntaktickej analýzy, keďže sú každá vykonávaná samostatne v prevodníkoch. Tento prevodník bol implementovaný a výsledná aplikácia je schopná preložiť zdrojový súbor v jazyku AIDA na medzikód AIDAcode, pričom v prípade chybného vstupu ohlási problém užívateľovi.

Do budúcna by bolo možné implementáciu LST prevodníka rozšíriť o použitie syntaktickej analýzy zdola-nahor pomocou LR gramatiky, keďže LR gramatika je silnejšia a schopnejšia akceptovať väčšie gramatiky. Užívateľ by mal možnosť vybrať si, ktorým spôsobom chce zdrojový súbor analyzovať, prípadne použiť oba a porovnať ich medzi sebou. Taktiež by implementovanie grafického uživateľského prostredia pomohlo prehľadnosti a intuitívnejšiemu použitiu výslednej aplikácie.

Literatúra

[1] Aho, A. V., Lam, M. S., Sethi, R. a Ullman, J. D. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321486811.

[2] Aho, A. V. a Ullman, J. D. *The Theory of Parsing, Translation, and Compiling*. USA: Prentice-Hall, Inc., 1972. ISBN 0139145567.

[3] Křivka, Z., Lukáš, R. a Rychnovský, L. *Jak na projekt* [online]. Brno: [b.n.], 2007 [cit. 2021-15-04]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IFJ/private/projekt/jak_na_projekt_prirucka.pdf.

[4] Křivka, Z., Zobal, L. a Genčúrová Lubica. *ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL* [online]. Brno: [b.n.], 2019 [cit. 2021-15-04]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php?file=%2Fcourse%2FIFJ-IT%2Fprojects%2FHistory%2Fifj2019.pdf&cid=13305.

[5] Meduna, A. *Automata and Languages: Theory and Applications*. Berlin, Heidelberg: Springer-Verlag, 2000. ISBN 1852330740.

[6] Meduna, A. *Formal Languages and Computation: Models and Their Applications*. 1st. USA: Auerbach Publications, 2014. ISBN 1466513454.

[7] Meduna, A. a Lukáš, R. *Formální jazyky a překladače IFJ Studijní opora* [online]. Brno: [b.n.], 2006 [cit. 2021-15-04]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php?file=%2Fcourse%2FIFJ-IT%2Ftexts%2FOporaIFJ.pdf&cid=13305.

[8] Rozenberg, G. a Salomaa, A., ed. *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Berlin, Heidelberg: Springer-Verlag, 1997. ISBN 3540604200.

[9] Techet, J., Masopust, T. a Meduna, A. *Parallel Communicating Grammar Systems* [online]. Brno: [b.n.], 2007 [cit. 2021-15-04]. Dostupné z: http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:10-pcgspres.pdf.

[10] Wilkin, M. a Bruda, S. Parallel communicating grammar systems with context-free components are Turing complete for any communication model. *Acta Universitatis Sapientiae, Informatica*. Október 2016, zv. 8. DOI: 10.1515/ausi-2016-0007.

Príloha A

LL gramatika

1.) <program> -> EOF
2.) <program> -> DEF ID (<param_group>):<command_end> <program>
3.) <program> -> IF (<expression>): <command> <else_expression> <program>
4.) <program> -> WHILE (<expression>): <command> <program>
5.) <program> -> FOR (<assing>;<expression>;<expression>):<command><program>
6.) <program> -> PASS; <program>
7.) <program> -> PRINT (<conversion_type> <param_group>); <program>
8.) <program> -> ID = <statement> <program>
9.) <program> -> <statement> <program>
10.) <param_group> -> ID <param>
11.) <param_group> -> {ϵ}
12.) <param> -> , ID <param>
13.) <param> -> {ϵ}
14.) <command> -> IF (<expression>): <command> <else_expression> <command_end>
15.) <command> -> WHILE (<expression >):<command> <command_end>
16.) <command> -> FOR(<assing>;<expression>;<expression>):<command>;<command_end>
17.) <command> -> PASS; <command_end>
18.) <command> -> PRINT (<conversion_type> <param_group>); <command_end>
19.) <command> -> ID = <statement>; <command_end>
20.) <command> -> <statement>; <command_end>
21.) <command> -> RETURN <statement>; <command_end>
22.) <command_end> -> <command>
23.) <command_end> -> {ϵ}
24.) <else_expression> -> else <command>
25.) <assign> -> ID =
26.) <statement> -> <conversion_type> <expression>;
27.) <statement> -> <func>
28.) <func> -> CHR (<conversion_type> <param_group>);
29.) <func> -> ORD (<conversion_type> <param_group>);
30.) <func> -> SUBSTR (<conversion_type> <param_group>);
31.) <func> ->LEN (<conversion_type> <param_group>);
32.) <func> -> INPUTS();
33.) <func> -> INPUTI();
34.) <func> -> INPUTF();
35.) <func> -> INPUTB();

36.) <conversion_type> -> INT
37.) <conversion_type> -> BOOLEAN
38.) <conversion_type> -> STRING
39.) <conversion_type> -> DOUBLE
40.) <conversion_type> -> {ϵ}

Príloha B

Stavový automat